

APPENDIX A

Smart Cameras as Embedded Systems

Smart cameras capture high-level descriptions of a scene and perform real-time analysis of what they see. These low cost, low-power systems push the design space in many dimensions, making them a leading-edge application for embedded system research.

Wayne Wolf
Burak Ozer
Tiehan Lu
Princeton University

Increasingly powerful integrated circuits are making an entire range of new applications possible. Complementary metal-oxide semiconductor (CMOS) sensors, for example, have made the digital camera a commonplace consumer item. These light-sensitive chips, positioned where film would normally be, capture images as reusable digital files that users can upload to their computer, manipulate with software, and distribute electronically.

Recent technological advances are enabling a new generation of *smart cameras* that represent a quantum leap in sophistication. While today's digital cameras capture images, smart cameras capture high-level descriptions of the scene and analyze what they see. These devices could support a wide variety of applications including human and animal detection, surveillance, motion analysis, and facial identification.

Video processing has an insatiable demand for real-time performance. Fortunately, Moore's law provides an increasing pool of available computing power to apply to real-time analysis. Smart cameras leverage very large-scale integration (VLSI) to provide such analysis in a low-cost, low-power system with substantial memory. Moving well beyond pixel processing and compression, these systems run a wide range of algorithms to extract meaning from streaming video.

Because they push the design space in so many dimensions, smart cameras are a leading-edge application for embedded system research. The Embedded Systems Group in Princeton University's Department of Electrical Engineering (<http://www.ee.princeton.edu/~wolf/embedded-group/>) has developed a first-generation smart camera system that can detect people and analyze their movement in real time.

DETECTION AND RECOGNITION ALGORITHMS

Although there are many approaches to real-time video analysis, we chose to focus initially on human gesture recognition—identifying whether a subject is walking, standing, waving his arms, and so on. Because much work remains to be done on this problem, we sought to design an embedded system that can incorporate future algorithms as well as use those we created exclusively for this application.

As Figure 1 shows, our algorithms use both low-level and high-level processing. The low-level component identifies different body parts and categorizes their movement in simple terms. The high-level component, which is application-dependent, uses this information to recognize each body part's action and the person's overall activity based on scenario parameters.

Low-level processing

The system captures images from the video input, which can be either uncompressed or compressed (MPEG and motion JPEG), and applies four different algorithms to detect and identify human body parts.

Region extraction. The first algorithm transforms the pixels of an image, like that shown in Figure 2a, into an $M \times N$ bitmap and eliminates the background. It then detects the body part's skin area using a YUV color model with chrominance values downsampled by a factor of two. Next, as Figure 2b illustrates, the algorithm hierarchically segments the frame into skin-tone and non-skin-tone regions by extracting foreground regions adjacent to detected skin areas and combining these segments in a meaningful way.

Contour following. The next step in the process, shown in Figure 2c, involves linking the separate

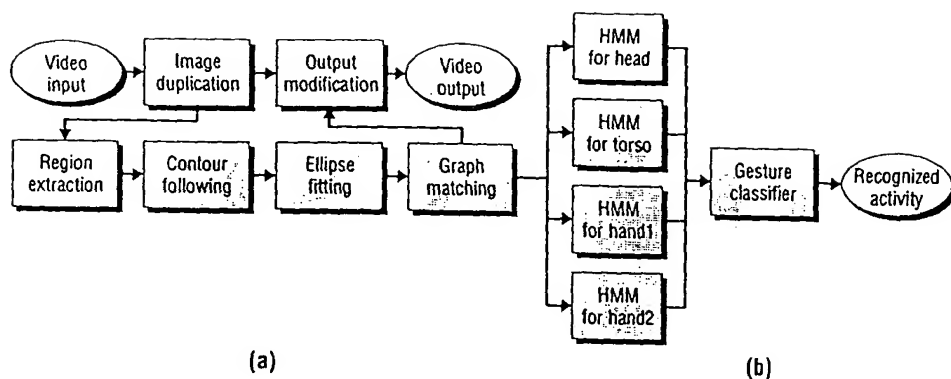


Figure 1. Human detection and activity recognition algorithms. (a) Low-level processing algorithms identify body parts and categorize their movements. (b) High-level processing algorithms use hidden Markov models (HMMs) and a gesture classifier to evaluate overall activity.

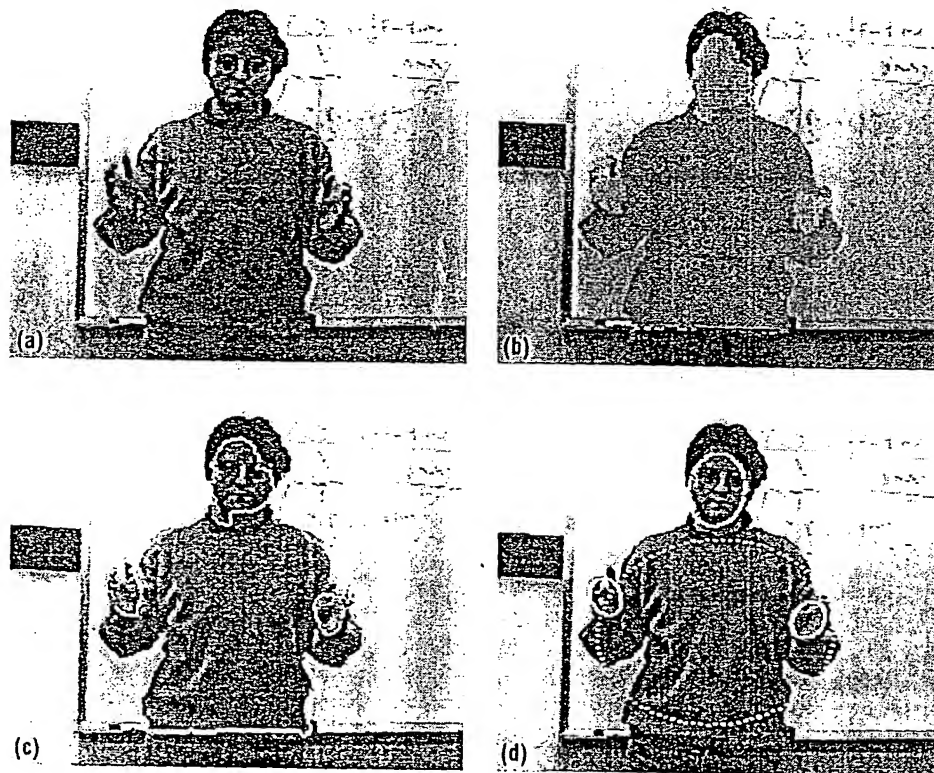


Figure 2. Initial steps in gesture recognition: (a) original image, (b) region extraction, (c) contour following, and (d) ellipse fitting.

groups of pixels into contours that geometrically define the regions. This algorithm uses a 3×3 filter to follow the edge of the component in any of eight different directions.

Ellipse fitting. To correct for deformations in image processing caused by clothing, objects in the frame, or some body parts blocking others, an algorithm fits ellipses to the pixel regions as Figure 2d shows to provide simplified part attributes. The algorithm uses these parametric surface approximations to compute geometric descriptors for segments such

as area, compactness (circularity), weak perspective invariants, and spatial relationships.

Graph matching. Each extracted region modeled with ellipses corresponds to a node in a graphical representation of the human body. A piecewise quadratic Bayesian classifier uses the ellipses parameters to compute feature vectors consisting of binary and unary attributes. It then matches these attributes to feature vectors of body parts or meaningful combinations of parts that are computed offline. To expedite the branching process, the algorithm

Motion-Detection and Gesture-Recognition Systems

The research efforts focusing on human motion detection and gesture-recognition systems include Leonard¹, a single-camera system that classifies simple motion events such as picking up an object using force dynamics. Mark Lucente, Gert-Jan Zwart, and Andrew D. George² have implemented a multimodal input system that also relies on one camera to let subjects manipulate virtual objects using gestures and voice commands. This system processes approximately 10 frames per second with a latency of 0.2 seconds.

The University of Maryland's Keck Laboratory for the Analysis of Visual Motion (<http://www.umi.acs.umd.edu/users/lzd/kecklab.html>) employs a multicamera system to construct dynamic graphical representations of human movement and object manipulation. Digital cameras simultaneously capture some activity—such as a technician repairing a mechanism—from multiple viewpoints, and a suite of networked computers integrates this data with other sensor information into a 3D model for analysis using advanced computer graphics. Thomas B. Moeslund and Erik Granum discuss related work in their survey.³

Mircea Nicolescu and Gérard G. Medioni⁴ have developed algorithms to electronically pan, tilt, and zoom through images supplied by an array of cam-

eras. Their qualitative criteria for evaluating video input have confirmed that pan-tilt-zoom systems outperform wide-angle-lens cameras. Jonathan Foote and Don Kimber⁵ have built a computationally and materially inexpensive panoramic camera system that also uses multiple cameras.

The MIT Media Lab (<http://www.media.mit.edu/>) is developing technology that can track people's actions, interpret gestures, and recognize facial expressions in environments ranging from the home and workplace to car interiors.⁶⁻⁹ Smart rooms, smart desks, and wearable computers use context-sensing and communication devices to unobtrusively help people carry out everyday functions.

Scott Stillman and Irfan Essa¹⁰ also have proposed a near-real-time system consisting of various types of sensors spread throughout an environment to track persons, detect faces, and recognize speech signals.

Other research efforts focus on various aspects of multiprocessor systems for video processing. Sek M.-Chai and colleagues, for example, have developed an architecture for pixel-level processing in the imaging array.¹¹ In addition, John A. Watlington and V. Michael Bove are building a dynamically scheduled dataflow system using a distributed

begins with the face, which is generally easiest to detect.

High-level processing

The high-level processing component, which can be adapted to different applications, compares the motion pattern of each body part—described as a spatiotemporal sequence of feature vectors—in a set of frames to the patterns of known postures and gestures and then uses several hidden Markov models in parallel to evaluate the body's overall activity. We use discrete HMMs that can generate eight directional code words that check the up, down, left, right, and circular movement of each body part.

Human actions often involve a complex series of movements. We therefore combine each body part's motion pattern with the one immediately following it to generate a new pattern. Using dynamic programming, we calculate the probabilities for the original and combined patterns to identify what the person is doing. Gaps between gestures help indicate the beginning and end of discrete actions.

A quadratic Mahalanobis distance classifier combines HMM output with different weights to generate reference models for various gestures. For example, a pointing gesture could be recognized as a command to "go to the next slide" in a smart

meeting room or "open the window" in a smart car, whereas a smart security camera might interpret the gesture as suspicious or threatening.

To help compensate for occlusion and other image-processing problems, we use two cameras set at a 90-degree angle to each other to capture the best view of the face and other key body parts. We can use high-level information acquired through one view to switch cameras to activate the recognition algorithms using the second camera. Certain actions, such as turning to face another direction or executing a predefined gesture, can also trigger the system to change views.

TOWARD AN EMBEDDED SYSTEM

As the "Motion-Detection and Gesture-Recognition" sidebar describes, a number of researchers are working on human motion detection and gesture-recognition systems.

We initially developed our gesture-recognition algorithms using Matlab (<http://www.mathworks.com/>). This technical computation and visualization programming environment runs orders of magnitude more slowly than embedded platform implementations, a speed difference that becomes critical when processing video in real time. We therefore ported our Matlab implementation to C code running on a very long instruction word

resource manager for compact, relatively inexpensive media-processing applications.¹²

References

1. J.M. Siskind, "Visual Event Classification via Force Dynamics," *Proc. 17th Nat'l Conf. Artificial Intelligence and 12th Conf. Innovative Applications of Artificial Intelligence (AAAI/IAAI 00)*, AAAI Press/MIT Press, Menlo Park, Calif., 2000, pp. 149-155.
2. M. Lucente, G.-J. Zwart, and A.D. George, "Visualization Space: A Testbed for Deviceless Multimodal User Interface," *Computer Graphics*, vol. 31, no. 2, 1997; <http://www.lucente.biz/pubs/pubs.html>.
3. T.B. Moeslund and E. Granum, "A Survey of Computer Vision-Based Human Motion Capture," *Computer Vision and Image Understanding*, vol. 81, no. 3, 2001, pp. 231-268.
4. M. Nicolescu and G.G. Medioni, "Electronic Pan-Tilt-Zoom: A Solution for Intelligent Room Systems," *Proc. IEEE Int'l Conf. Multimedia and Expo (ICME 00)*, IEEE CS Press, Los Alamitos, Calif., 2000, pp. 1581-1584.
5. J. Foote and D. Kimber, "FlyCam: Practical Panoramic Video and Automatic Camera Control," *Proc. IEEE Int'l Conf. Multimedia and Expo (ICME 00)*, IEEE CS Press, Los Alamitos, Calif., 2000, pp. 1419-1422.
6. A. Pentland and T. Choudhury, "Face Recognition for Smart Environments," *Computer*, Feb. 2000, pp. 50-55.
7. A.D. Wilson and A.F. Bobick, "Realtime Online Adaptive Gesture Recognition," *Proc. 15th Int'l Conf. Pattern Recognition (ICPR 00)*, IEEE CS Press, Los Alamitos, Calif., 2000, pp. 270-275.
8. A. Pentland, "Smart Rooms, Smart Clothes," *Proc. 14th Int'l Conf. Pattern Recognition (ICPR 98)*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 949-953.
9. A. Pentland, "Looking at People: Sensing for Ubiquitous and Wearable Computing," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, 2000, pp. 107-119.
10. S. Stillman and I. Essa, "Towards Reliable Multimodal Sensing in Aware Environments," *Proc. Workshop on Perceptive User Interfaces (PUI 01)*, ACM Press, New York, 2001; <http://www.cs.ucsb.edu/PUI/PUIWorkshop/>.
11. S.M. Chai et al., "Focal Plane Processing Architectures for Real-Time Hyperspectral Image Processing," *J. Applied Optics: Special Issue on Information Processing*, vol. 39, no. 5, 2000, pp. 835-849.
12. J. Watlington and V.M. Bove Jr., "A System for Parallel Media Processing," *Parallel Computing*, vol. 23, no. 12, 1997, pp. 1793-1809.

(VLIW) video processor, which let us make many architectural measurements on the application and make the necessary optimizations to architect a custom VLSI smart camera.

Requirements

At the development stage, we evaluated the algorithms according to accuracy and other familiar standards. However, an embedded system has additional real-time requirements:

- **Frame rate.** The system must process a certain amount of frames per second to properly analyze motion and provide useful results. The algorithms we use as well as the platform's computational power determine the achievable frame rate, which can be extremely high in some systems.
- **Latency.** The amount of time it takes to produce a result for a frame is also important because smart cameras will likely be used in closed-loop control systems, where high latency makes it difficult to initiate events in a timely fashion based on action in the video field.

Moving to an embedded platform also meant that we had to conserve memory. Looking ahead to highly integrated smart cameras, we wanted to

incorporate as little memory in the system as possible to save on both chip area and power consumption. Gratuitous use of memory also often points to inefficient implementation.

Components

Our development strategy called for leveraging off-the-shelf components to process video from a standard source in real time, debug algorithms and programs, and connect multiple smart cameras in a networked system. We use the 100-MHz Philips TriMedia TM-1300 as our video processor. This 32-bit fixed- and floating-point VLIW processor features a dedicated image coprocessor, a variable length decoder, an optimizing C/C++ compiler, integrated peripherals for concurrent real-time input/output, and a rich set of application library functions including MPEG, motion JPEG, and 2D text and graphics.

Our testbed architecture, shown in Figure 3, uses two TriMedia boards attached to a host PC for programming support. Each PCI bus board is connected to a Hi8 camera that provides NTSC composite video. Several boards can be plugged into a single computer for simultaneous video operations. The shared memory interface offers higher performance than the networks likely to be used in VLSI cameras, but they let us functionally imple-

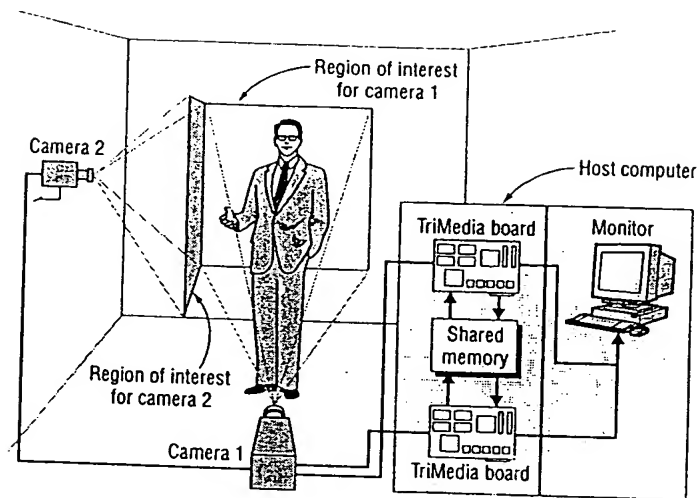


Figure 3. Smart camera test room and testbed architecture. Principal components include video processors on standard PCI bus cards, a shared memory interface, and a host PC for programming support.

ment and debug multiple-camera systems with real video data.

EXPERIMENTS AND OPTIMIZATIONS

After converting the original Matlab implementation into C, we performed some experiments to gauge the smart camera system's effectiveness and evaluate bottlenecks. The unoptimized code took, on average, 20.4 million cycles to process one input frame, equal to a rate of 5 frames per second.

We first measured the CPU times of each low-level processing step to determine where the cycles were being spent. Microsoft Visual C++ is more suitable for this purpose than the TriMedia compiler because it can collect the running time of each function as well as its subfunctions' times.

Figure 4a shows the processing time distribution of the four body-part-detection algorithms. Figure 4b shows the memory characteristics of each low-level processing stage.

As data representation becomes more abstract, input/output data volume decreases. The change in required memory size, however, is less predictable given the complex relationships that can form between abstract data. For example, using six single-precision, floating-point parameters to describe 100 ellipses requires only 2.4 Kbytes of memory, but it takes 10 Kbytes to store information about two adjoining ellipses.

Based on these early experiments, we optimized our smart camera implementation by applying techniques to speed up video operations such as substituting new algorithms better suited to real-time processing and using TriMedia library routines to replace C-level code.

Algorithmic changes

We originally fit superellipses (generalized ellipses) to contour points, and this was the most time-consuming step. Rather than trying to optimize the code, we decided to use a different algo-

rithm. By replacing the original method developed from principal component analysis with moment-based initialization, we reduced the Levenberg-Marquardt fitting procedure, thus decreasing the execution time. Also, to accelerate processing during the graph-matching stage, we modified the algorithm to determine different regions' adjacency.

Library functions

During the region-extraction stage, the system processes each pixel in the input frame independently. Absolute value and threshold calculations result in branching, which limits instruction-level parallelism (ILP). One possible solution to this problem is to split the frame into several pieces and process these pieces on a multiprocessor or simultaneous multithreading platform. However, we opted to reduce the number of branches in the program.

The TriMedia processor provides an INONZERO operation that takes two input operands. If the first is not zero, the destination is set to the value of the second operand; otherwise, it is set to zero. Another special operation, IABS, can provide absolute values. These operations are visible in C code as they are packed into functions, and together they remove most of the branches.

We also used loop unrolling to extend basic block size. This optimization increased the processing speed of the region-extraction step by a factor of 2.3.

Control-to-data transformation

Increasing the processor's issue width can exploit the high degree of parallelism that region extraction offers. Using a processor with more functional units could thus reduce processing time during this stage. However, contour following, which converts pixels to abstract forms such as lines and ellipses, consumes even more time. The algorithm also operates serially: It finds a region's boundary by looking at a small window of pixels and sequentially moving around the contour; at each clockwise step it must evaluate where to locate the contour's next pixel. While this approach is correct and intuitive, it provides limited ILP.

We evaluated all possible directions in parallel and combined the true/false results into a byte, which served as an index to look up the boundary pixel in a table. We also manipulated the algorithm's control-flow structure to further increase ILP. These optimizations doubled the contour-following stage's running speed.

Optimization results

The combination of these methods radically

improved CPU performance for the application. Optimization boosted the program's frame rate from 5 to 31 frames per second. In addition, latency decreased from about 340 to 40-60 milliseconds per frame. We have since added HMMs and other high-level processing parts, and the program now runs at about 2.5 frames per second.

Our board-level system is a critical first step in the design of a highly integrated smart camera. Although the current system is directly useful for some applications, including security and medicine, a VLSI system will enable the development of high-volume, embedded computing products.

Because the digital processors and memory use advanced small-feature fabrication and the sensor requires relatively large pixels to efficiently collect light, it makes sense to design the system as two chips and house them in a multichip module. Separating the sensor and the processor also makes sense at the architectural level given the well-understood and simple interface between the sensor and the computation engine.

We believe that embedding single-instruction multiple-data (SIMD) processors into the sensor is not critical to achieve real-time performance. The advantages of leveraging existing sensor technology far outweigh any benefits of using pixel-plane processors until they become more plentiful. However, attaching special-purpose SIMD processors to the multiprocessor can be useful for boundary analysis and other operations. Such accelerators can also save power, which is important given the cost and effort required to deploy multiple cameras, especially in an outdoor setting. High-frame-rate cameras, which are useful for applications ranging from vibration analysis to machinery design, will likely require many specialized processing elements that are fast as well as area efficient, allowing the inclusion of more parallel units.

We are still in the early stages of determining the type of network best suited to a multicamera system. Distributed processing is an important means of reducing power consumption in such systems—sending raw pixels over the network is less efficient than sending the results of intermediate analysis. However, as algorithms for real-time multicamera analysis continue to develop, bandwidth requirements may change.

Acknowledgments

This work was funded by the New Jersey Center

for Pervasive Information Technology; the MARCO/DARPA Center for Circuits, Systems, Software; and the National Science Foundation. We also thank Kees Vissers, Flaviu Turean, and Sebastian Mirolo at TriMedia, which generously provided us with processor boards, for their constructive suggestions.

Wayne Wolf is a professor in Princeton University's Department of Electrical Engineering, where he heads the Embedded Systems Group, and an associated faculty member in the Department of Computer Science. He received a PhD in electrical engineering from Stanford University. He is a Fellow of the IEEE and the ACM. Contact him at wwolf@princeton.edu.

Burak Ozer is a research staff member and member of the Embedded Systems Group in the Department of Computer Engineering at Princeton University. He received a PhD in electrical engineering from the New Jersey Institute of Technology. He is a member of the IEEE. Contact him at iozer@ee.princeton.edu.

Tiehan Lu is a graduate student and member of the Embedded Systems Group in the Department of Electrical Engineering at Princeton University. He received an MAE in electrical engineering from Peking University, China. Contact him at lv@ee.princeton.edu.

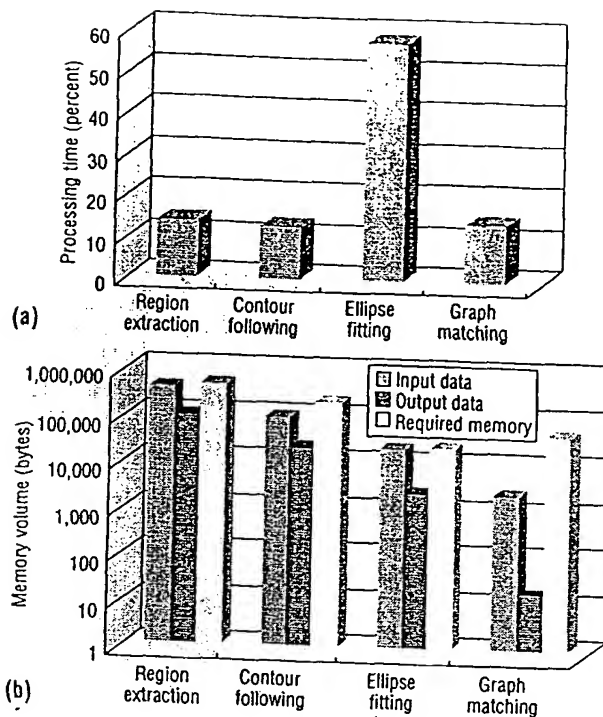


Figure 4. (a) Processing time distribution of the original algorithms, implemented in C, before optimization. (b) Memory volume of low-level processing stages.

A BOTTOM-UP APPROACH FOR ACTIVITY RECOGNITION IN SMART ROOMS

Burak Ozer, Tiehan Lv and Wayne Wolf

Department of Electrical Engineering, Princeton University, Princeton, NJ 08544, USA
{iozer,lv,wolf}@ee.princeton.edu

ABSTRACT

In this paper, we propose a smart camera system where the cameras detect the presence of a person and recognize activities of this person. A relational graph-based modeling of human body and a HMM-based activity recognition of the body parts are proposed for real-time video analysis. The results show that more than 86 percent of the body parts and 88 percent of the activities are correctly classified. We also describe the relationship between the activity detection algorithms and the architectures required to perform these tasks in real time. We achieve a processing rate of more than 20 frames per second for each TriMedia video capture board.

1. INTRODUCTION

The increased importance of applications requiring fast, cheap, small and highly accurate smart cameras, such as surveillance systems, necessitates research efforts to provide efficient solutions to the problem of real-time detection of persons and classification of their activities. Smart cameras create smart environments, such as cars that can recognize driver's gestures, rooms with multimedia devices that can be controlled by speech or gesture, or security checkpoints where suspicious movements and unwanted people are detected. We have developed algorithms for real-time detection of persons and classification of their activities in a room equipped with multiple-cameras. Human detection is achieved via graph matching of nodes that represent approximated shapes of detected body parts. We then compare the constellation of shapes against models in a library. The activity of each body part is recognized by a HMM-based activity/gesture recognition algorithm and the outputs of the HMMs are combined to generate scenarios for the overall body activity.

The paper describes our work for online activity detection by using Trimedia cards that also enable to record the sequences in MPEG format for off-line video analysis purposes [1]. Two important contributions are presented in this paper. First, we test and improve the performance of the human detection algorithm steps [2] in terms of accuracy and computational complexity by using our new testbed system with VLIW processors for video operations. This platform allows us to evaluate algorithms for our long-term goal to develop an integrated smart camera that includes a sensor, on-board processing, and on-board memory. Second, we propose a HMM based activity recognition for on-line video analysis. The paper is organized as follows. Section 2 is a review of existing literature devoted to smart rooms and smart cameras. In Section 3, we describe the algorithm for detection of the human body parts. In Section 4 we propose an HMM-based activity recognition. The experimental results are presented in Section 5. Based on the algorithmic overview, Section 6 describes how the algorithms affect the underlying architecture and briefly describes our PC-based testbed. Conclusions are presented in Section 7.

2. PREVIOUS WORK

An extensive research has been done at MIT Media Lab on smart rooms for several applications [3] where computer systems that can

follow people's actions, recognizing their faces, gestures, and expressions are developed. A similar research is conducted in the University of Maryland Keck Laboratory for the analysis of visual motion where multiple cameras are attached to a network of sixteen PCs used for both data collection and real time video analysis [4]. Several other groups have developed multi-camera and multi-processor systems for video processing [5, 6]. Most of the previous work for human recognition which is an important application for smart rooms depend highly on the segmentation results and mostly motion is used as the cue for segmentation [7]. Major approaches for analyzing human activity patterns include Dynamic Time Warping (DTW) [8], Neural Networks (NNs) [9], and Hidden Markov Models (HMMs) [10]. Most of the activity/gesture detection techniques rely on the successful feature extraction and proposed approaches for smart cameras are suitable for a specific application type. Our aim is to develop a smart camera system that can detect a wide range of activities for different applications. For this reason, our scheme detects different body parts and their movement in order to combine them at a later stage that connects to high-level semantics. Our scheme uses model-based segmentation [2] where high-level nodes are created from the combination of low-level segments via a feedback from graph matching part. In this paper, we optimize our detection algorithm for our new testbed system and we propose a HMM based activity detection scheme that uses the same bottom-up approach to combine individual movements of the body parts.

3. LOW-LEVEL PROCESSING

This section presents the proposed algorithm for the detection of the human body parts. The algorithm blocks are displayed in Figure 1. A more detailed explanation of the algorithm can be found in [2].

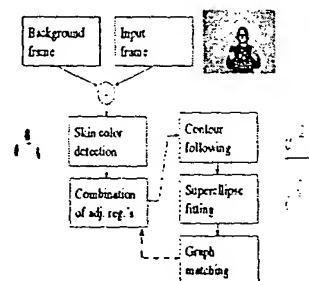


Figure 1: Algorithm blocks and corresponding results of selected steps.

Background elimination and color transformation: First step is the transformation of pixels into another color space regarding to the application. Background elimination is performed by using these transformed pixel values for the current and background images. Skin area detection: Skin areas are detected by comparing color values to a human skin model. We use YUV color model where chromi-

nance values are downsampled by two.

Segmentation of non-skin areas and connected component algorithm: The foreground regions that are adjacent to detected skin areas are extracted and corresponding connected components are found. We combine the meaningful adjacent segments and use them as the input of the following algorithm steps.

Contour following: We apply the contour following algorithm that uses the 3x3 filter to follow the edge of the component where the filter can move in any of 8 directions to follow the edge.

Superellipse fitting: 2D approximation of body parts by fitting superellipses with shape preserving deformations helps to overcome the occlusion problem. Each contour of size c_i is then fitted to a superellipse with 5 parameters by a Levenberg-Marquardt minimization method.

Object modeling by invariant shape attributes: The descriptors of body parts are classified into two groups: unary (compactness, eccentricity) and binary (ratio of areas, relative position and orientation, adjacency) attributes.

Graph matching: Each extracted region modeled with ellipses correspond to a node in the graphical representation of human body. Face detection allows to start initial branches efficiently and reduces the complexity. Each body part and meaningful combinations represent a class (ω) where the combination of binary and unary features are represented by a feature vector (X) that is computed off-line. For the purpose of determining the class of the feature vectors a piecewise quadratic Bayesian classifier with discriminant function $g(X)$ is used. The generality of the reference model attributes allows the detection of different postures while the conditional rule generation (r) decreases the rate of false alarms.

Robust detection of the person(s) in the scene through occlusions requires multiple cameras. Our assumption is that human face (at least a part of it) must be seen since skin color is a dominant attribute for head, reducing the complexity of the detection algorithm. We choose the camera with the best view angle which can capture frontal or near frontal view of the face. Once the face is detected after graph-matching step, its attributes are used as the input of the camera switching algorithm.

4. HIGH-LEVEL PROCESSING

This section covers the proposed real-time activity recognition algorithm based on Hidden Markov Models (HMMs). HMM is a statistical modeling tool that helps to analyze time varying signals. Online handwriting recognition [12], video classification and speech recognition [13] are some of the application areas of HMMs. Only a few researchers have used the HMM to recognize activities of the body parts. It is mainly used for hand gestures [10]. Parameterized-HMM [14] can recognize complex events such as an interaction of two mobile objects, gestures made with two hands (e.g. so big, so small), etc.. One of the drawbacks of the parameterized HMM is that for complex events (e.g. a combination of sub-events) parameter training space may become very large. In our application, we assume that each body part has its own freedom of motion and the activity recognition for each part is achieved by using several HMMs in parallel. Combining the outputs of the HMMs to generate scenarios is an application dependent issue. In our application environment, smart room, we use Mahalanobis distance classifier for combining the activities of different body parts by assigning different weights for each activity. An HMM can be represented by using the notation $\lambda = (A, B, \pi)$ [11] where A , B , and π represent the transition, output, and initial probabilities, respectively. The movement of the body parts are described as a spatio-temporal sequence of feature vectors that consist of the direction of the body part movement. Since we use discrete HMMs, we generate 8 directional codewords. We check the up, down, right, left, and circular movements of the body parts. Our models are trained using Baum-Welch algorithm. Note that the

detected movement of the body part may be a part of a more complex activity. We check the current pattern and combine it with the immediately following one and generate a new pattern. Using dynamic programming we calculate the probabilities for the first and combined patterns and choose the pattern with the highest probability as the recognized activity. If the probability of the observed activity is below a threshold, we reject the activity. Furthermore, we use the gap between different gestures/activities, e.g., moving the hand out of camera, stopping the body for a while. Another feature in the activity recognition is the speed of the body parts. We use the speed of each body part (slow/fast) for one activity period as an additional input for the classification. Next step is the generation of a feature vector by using the observed activities of the body parts. The activity feature vector is compared with the known activities via a distance classifier, based on Mahalanobis metric. The output of the classifier detects the overall activity of the person. Proposed activity classification algorithm is given in Figure 2.

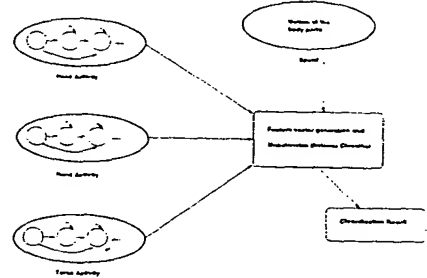


Figure 2: Overview of the activity classification. The quantity r :

$$r^2 = (x - m_x)' C_x^{-1} (x - m_x) \quad (1)$$

is called the Mahalanobis distance. x represents the feature vector where each element of the vector corresponds to the activity of different body parts during the observation period. Note that different body parts have different activity periods but the observation period is the same for each of them. The activity of each body part are written to the feature vector. m_x is the mean vector of each class. C_x is the covariance matrix for x . We classify a feature vector x by measuring the Mahalanobis distance from x to each of the means, and assigning x to the class for which the Mahalanobis distance is minimum. Weighted mean and weighted covariance matrix during the classification step help to give different weights to the recognized activity of each body part.

Our smart room has two cameras that are located by 90 degrees to each other. Besides the low level information explained in section 3, we also use high level information, namely output of the activity recognition algorithm, for camera switching purposes. If we recognize a specific gesture of the person (pointing to the second camera by using his/her hand without moving his/her head and torso), we switch the cameras. Note that the Trimedia processors and the host PC communicate through shared memory. Due to the false detection of the activities as well as the low level parameters, there is a possibility of rapid change of the cameras. To solve this problem, we continue the process on one card for at least 100 frames.

5. RESULTS

In this section, we present results for different steps of the algorithm for performance evaluation. Some snapshots of the sequence for each algorithm block are displayed in Figure 3. 86% of the body parts in the processed frames are correctly classified, the rest is the miss and false classification.

Some of the example frames from different videos for the recognized activity sequences are displayed in Figure 4. Some of the



Figure 3: Snapshot of a sequence for each algorithm block. Left: Skin color detection. Middle: Contour following. Right: Superellipse fitting.

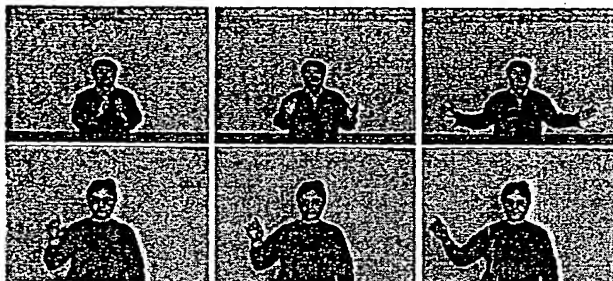


Figure 4: Example frames from the recognized activity sequence: Top: Opening and closing arms, Bottom: Pointing door

activities are opening and closing arms, pointing camera and door, walking to the door and camera, moving up and down, circling hands (clockwise/counter-clockwise), etc. (Figure 5). Note that for each activity pattern of the body parts we use 30 training sequences as the input of the Baum-Welch algorithm. The process of constructing complex patterns by combining the sub-patterns of the body parts is achieved by the distance classifier. Activity recognition results for some of the example patterns are given in Table 1. Most of the false detections of the exemplar activities occur because of the classification errors of the body parts. Note that, smaller body parts, such as hands and fingers, have higher false classification ratios than the larger ones.

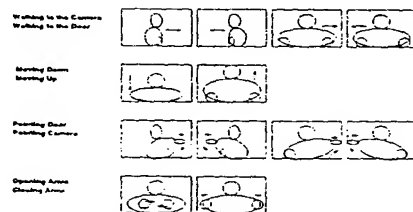


Figure 5: Some example patterns.

Figure 6 (top) displays fitted superellipses for some of the activity patterns, namely waving one hand, opening and closing hands, and left-right movement of the body. It displays cumulative motion of the body parts. In Figure 6 (bottom), circling-hand activity pattern is displayed. We observe that different activity patterns can have overlapping periods (same or similar patterns for a period) for some body parts. Hence the detection of start and end times of activities is crucial. As stated earlier, to detect the start and end time of a gesture, we use the gap between different gestures/activities. We also check the current pattern and combine it with the immediately following one and generate a new pattern. Furthermore, the speed of the body parts is used as another feature in the activity recognition. Note that the combination of the activities of the body parts make this detection more reliable. For example, consider a person who, after pointing toward an object (head and torso are idle while hand points out), starts walking to the door (all body parts move toward door, and hands are moved down). Although body parts have different importance in activity patterns, all the body parts' activities contribute with different weights to the detection of the start and end

Pattern	Correct	False
Walking to the Camera	9	0
Walking to the Door	9	0
Turning to the Camera	10	1
Turning to the Door	10	2
Pointing Camera	14	3
Pointing Door	15	2
Open Hands	11	2
Close Hands	11	2
Circling Hands	8	2
Moving Up	9	0
Moving Down	8	1

Table 1: The results for activity classification.

Float Compare	Average Processing Speed (Cycles per Frame)
Low level part	2.78e+07
High level part	1.74e+05

Table 2: Average Processing Times

times of the overall activity pattern.

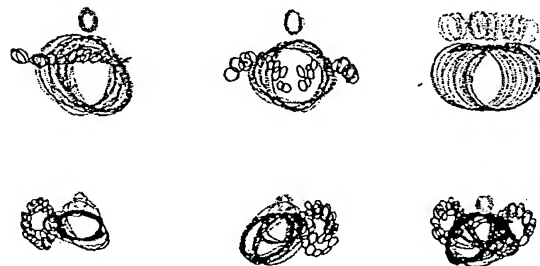


Figure 6: Cumulative motion of body parts (fitted superellipses) for different activity patterns: Top: Waving one hand, opening and closing hands, left-right movement, Bottom: Circling right hand, circling left hand, circling first right and then left hand.

6. IMPLEMENTATION ENVIRONMENT AND OPTIMIZATION

It is essential for us to understand the application behavior to develop efficient hardware for a smart camera system. To build our prototype system [15], we use a PC platform with two TriMedia Media Processing boards. In this section, we present the optimizations on the algorithm. Although the complicated modern compilers can perform quite efficient and complex optimization for applications, manual optimization on specific algorithms can still bring a considerable speedup. For these reasons, we perform manual optimization. After the optimization the running time is reduced by more than 80%.

Optimization of the Proposed Algorithm: As stated above, the algorithm has a low level processing part and a high level processing part. The low level part is processing on a frame basis, and the high level part takes the relationship between several consecutive frames as input. Therefore, it is clear that the low level part is the most time consuming part of the algorithm, which is also proved by our experimental results as presented in Table 2. For this reason, we focus our optimization on the low level part.

Algorithm Level Optimization: Algorithm level optimization replaces time-consuming code segment by code that is more efficient. It helps to deep understand of the target program. It is evident that the optimization should be performed on the most time consuming parts of the algorithm. The original time distribution for major functions in

Function	Execution Time (Milliseconds)	Percentage
RegionExtract	7.029	13.8%
ContourExtraction	6.668	13.1%
SuperFit	29.881	58.6%
Match	7.388	14.5%
Total	50.966	100%

Table 3: Function execution times of original program.

Function	Execution Time (Milliseconds)	Percentage
RegionExtract	6.954	48.2%
ContourExtraction	6.644	46.1%
SuperFit	0.111	0.8%
Match	0.710	4.9%
Total	14.419	100%

Table 4: Running time of the functions after optimization on Adjacent.

the algorithm is shown in Table 3. The super-ellipse fitting and graph matching are the most time consuming parts. By using moment-based initialization to replace the original method developed from Principle Component Analysis, we can remove Levenberg-Marquardt fitting procedure. Thus, we successfully cut down the execution time of the super-ellipse fitting and graph matching. After these optimizations, the total execution time of the algorithm dropped from 51 milliseconds to 14 milliseconds (Table 4). The tradeoff is between accuracy and computational complexity. Our current and future work include the selection of algorithms for different segments and nodes (e.g. superellipses for torso vs deformed superquadrics for fingers) by using a feedback from graph-matching and model-based segmentation.

Low Level Optimization for TriMedia Processor: Five issue VLIW TriMedia processors offer abundant resources for instruction level parallelism. The difference between the average basic block size of the algorithm and the instructions per cycle of the program suggest that manual optimization may reduce the running time dramatically. On a TriMedia processor, the processing times are listed in Table 5 where functions RegionExtract and ContourExtraction are the most time consuming parts. In the table, 'Others' means the remaining part of the program.

Using restricted pointers, loop unrolling and applying the custom functions offered by TriMedia, we eliminate the most part of branches in region extraction. The impact on running time is significant as shown in Table 6. In this step, total execution time is reduced by 33%. After these optimizations, the total processing time for a frame is $4.36 \times 10^6 + 1.74 \times 10^5 = 4.53 \times 10^6$ cycles. Given that the clock rate of the TriMedia Processor is 100MHz, one TriMedia processor can process more than 20 frames per second, which is more than enough for ordinary movements.

7. CONCLUSIONS

In this paper, we study the relationship between our algorithms proposed for human activity detection and the architectures required to perform these tasks in real time for a smart camera system. Our algorithms use graph models and Hidden Markov Models to detect people and describe their activity. We can achieve a correct classification

Function	Total Cycles (x1000)	Percentage
RegionExtract	2737	36.66%
ContourExtraction	3145	42.12%
Others	1588	21.22%
Total	7470	100%

Table 5: Function times before restricted pointers and loop unrolling.

Function	Total Cycles (x1000)	Percentage
RegionExtract	1518	34.84%
ContourExtraction	1720	39.49%
Others	1118	25.67%
Total	4356	100%

Table 6: Function times after optimization.

rate of 86% for the body parts and we can detect 88% of the activities correctly. We achieve a processing rate of more than 20 frames per second for each TriMedia video capture boards.

8. REFERENCES

- [1] B. Ozer, W. Wolf, "Human Detection in Compressed Domain", ICIP, Greece, 2001.
- [2] B. Ozer, W. Wolf, A. N. Akansu, "Relational Graph Matching for Human Detection and Posture Recognition", SPIE, Photonic East 2000, Internet Multimedia Management Systems, Boston, November 2000.
- [3] A. Pentland, T. Choudhury, "Face Recognition for Smart Environments", Computer, Vol. 33/2, pp. 50-55, Feb. 2000.
- [4] L. S. Davis, Eugene Borovikov, Ross Cutler, and Thanarat Horprasert, "Multi-perspective Analysis of Human Action", Third International Workshop on Cooperative Distributed Vision, 1999.
- [5] S. M. Chai, A. Gentile, W. E. Lugo-Beauchamp, J. Fonseca, J. L. Cruz-Rivera, and D. S. Wills, "Focal Plane Processing Architectures for Real-time Hyperspectral Image Processing," Applied Optics, Special Issue on Optics in Computing, 39:(5), pages 835-849, February 2000.
- [6] J. Watlington and V. M. Bove, Jr., "A System for Parallel Media Processing," Parallel Computing, 23:12, December 1997.
- [7] J. K. Aggarwal and Q. Cai, "Human Motion Analysis: A Review," Computer Vision and Image Understanding, vol. 73, no. 3, pp. 428-440, March 1999.
- [8] K. Takahashi, S. Seki, R. Oka, "Spotting Recognition of Human Gestures from Motion Images," Tech. Report IE92-134, 1992.
- [9] R. Kjeldsen and J. Kender, "Visual Hand Gesture Recognition for Window System Control", Proc. International Workshop Automatic Face and Gesture Recognition, pp. 184-188, Zurich, 1995.
- [10] T. Starner and A. Pentland, "Real-Time American Sign Language Recognition from Video Using Hidden Markov Models," Technical Report TR-375, MIT's Media Lab., 1995.
- [11] X.D. Huang, Y. Ariki, and M.A. Jack, "Hidden Markov Models for Speech Recognition," Edinburgh: Edinburgh Univ. Press, 1990.
- [12] B.K. Sim and J.H. Kim, "Ligature Modeling for Online Cursive Script Recognition," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 19, no. 6, pp. 623-633, June 1997.
- [13] R.C. Rose, "Discriminant Wordspotting Techniques for Rejection Non-Vocabulary Utterances in Unconstrained Speech," Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing, vol. II, pp. 105-108, San Francisco, 1992.
- [14] A. D. Wilson and A. F. Bobick, "Parametric Hidden Markov Models for Gesture Recognition," IEEE Trans. on PAMI, Vol. 21, No. 9, pp. 884-900, September 1999.
- [15] T. Lv, B. Ozer, and W. Wolf, "Workload Characterization for Smart Cameras", 3rd Workshop on Media and Streaming Processors, The 34th Symposium on Microarchitecture, Austin, December 2001.

Princeton, NJ 08544

Real-time Posture and Activity Recognition

Abstract

In this paper, we propose an algorithm for human posture and activity recognition for uncompressed and compressed (MPEG) video inputs. A real-time compression domain technique is developed to recognize different postures such as standing, pointing left/right, opening arms, etc. by using an eigenspace representation of human silhouettes obtained from AC-DCT coefficients. The system stores frames with specific postures and finds global activity of the human body in the compressed domain. In the uncompressed domain, this information is used as an input for the activity/gesture recognition algorithm. First part of our approach is invariant to changes in intensity, color and textures and has the advantage of using the available data in the standard compression algorithms. Second part of the system can recognize activities in a set of frames starting with a recognized posture that is classified as a reference movement by the system. A prototype system is developed with two camera nodes each consists of a standard camera and a video processing board.

1 Introduction

Recent advances in camera and storage systems are main factors driving the increased popularity of video surveillance [1]. Prices continue to drop on components such as CMOS cameras while manufacturers have added more features. Furthermore, the evolution of digital video especially in digital video storage and retrieval systems is another leading factor. Besides the expensive surveillance systems, today's PC-based, easy plug-in surveillance systems are directed at home users and small business owners who can not afford the expense of investing thousand of dollars for a security system. Real time monitoring from anywhere, anytime enable keeping a watchful eye on offices, stores, houses, pools or parking garages. This is achieved via remote monitoring from internet environment. Some systems send out predefined e-mail alerts with surveillance recordings attached or initiate phone call and local alarm when an activity is detected. This makes it possible to observe remotely whatever activity surveillance camera captures. Once recorded, surveillance videos are stored for later replay. During playback of the captured motion, there's no need to watch every single video frame since user can skip through motion events. In addition, advanced scheduling makes regular monitoring easy and automatic. The user can predefine timed or motion-triggered surveillance events. If the PC is connected to a network, then video files can be recorded onto a remote server, just in case something happens to the PC. Remote monitoring over internet and video storage require efficient compression techniques. For this reason, most of the systems use standard compression algorithms, such as MPEG or Motion JPEG dig-

ital video recording. Although these surveillance systems are powerful with new advances in camera and storage systems, activity detection and automatic alarm systems are not mature yet. These topics are still open areas for many research groups. Most of the research in this area are done for uncompressed domain. Although surveillance videos rely on standard compression techniques for efficient storage and transmission of video and image files, real-time compression domain techniques for activity recognition is not fully investigated for these systems.

The purpose of this work is to develop a real-time system for posture recognition and activity detection in the compressed domain. Our aim is to reduce computational complexity, to avoid dependency on correct segmentation, and to reduce storage area and bandwidth requirements. Furthermore, we show the interaction of this system with our activity recognition system in the uncompressed domain in order to recognize specific gestures for different type of applications, such as surveillance systems, smart environments, entertainment, etc.

In our previous work ([2] and [3]), we proposed an algorithm for human detection in JPEG compressed still images and MPEG I frames by using an eigenspace representation of human silhouettes for non-real-time applications, such as video and/or image libraries. In this paper, we modify and test our algorithm for real-time video inputs where we can recognize a) postures, b) motion direction of the human body, and c) gestures of a person by using the proposed hierarchical system shown in Figure 1. For this purpose, we use two Tri-Media video processing boards, that are specialized for media processing and can communicate via shared memory. Great effort has been devoted to human recognition related topics such as face recognition in still images, and motion analysis of human body parts. Most of the previous work depend highly on the segmentation results and mostly motion is used as the cue for segmentation. Gesture and activity detection are the classification of finite characteristic motions captured with cameras or other sensors. Hence gesture/activity detection involves the classification of spatio-temporal patterns. Most of the related work use activity measurements from uncompressed images after a proper segmentation of human body parts. These techniques rely on the successful feature extraction. Our scheme uses blocks in compressed domain and model-based segmentation in uncompressed domain where high-level nodes are created from the combination of low-level segments via a feedback from graph matching part.

The major contribution of the overall algorithm is to connect available data in low-resolution compressed domain to high level semantics. Consider a video stream taken from a fixed camera surveying a passage. If the system detects an activity it will start recording MPEG compressed video.

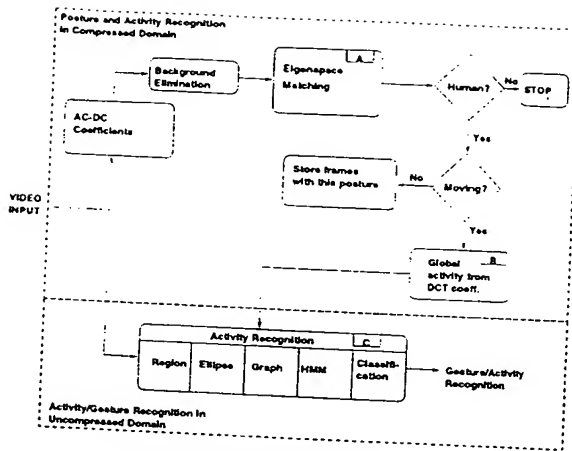


Figure 1. System overview.

The first step will retrieve possible frames in the compressed domain where people are present. The system will analyze the extracted region for posture recognition. If human posture is detected, the system will send an alarm or a message and check the motion direction of the human silhouette by using DCT differences, e.g. moving right, left, up, down, etc. If a suspicious movement is detected (such as pointing), the next step will be a more detailed investigation of the activity/gesture of the person in the uncompressed domain. Related work for posture and gesture/activity recognition are given in the following section. In Section III we describe details of the major algorithm blocks and the overall system and the PC based testbed architecture. We present the performance of the system in Section IV. Section V concludes the paper.

2 Related Systems

Photobook [4] project uses a compact eigenspace representation of faces that can be used for both recognition as well as image compression. In [5], the structural information of pedestrians is presented by a subset of wavelet coefficients and pedestrians are detected by the support vector machine classification method. Our previous work [2] aims to retrieve information from images and videos compressed using standard algorithms such as JPEG and MPEG. This differentiates our approach from the previous work where the compression algorithms are governed by characteristics of object of interest to be retrieved. In our algorithm, the overall shape of a standing or walking person (from front or back-view) in still images is detected by using the AC-DCT coefficients.

As the input of the system we can use DC and AC coefficients from MPEG frames or calculate DC and first order AC coefficients of each frame for real-time video streams.

The use of available information in compressed video and images has been investigated mostly for video indexing, and shot and scene classification. The object detection in the compressed domain is more restricted since this application requires more detailed information. Schonfeld [6] proposes an object tracking algorithm by using compressed video only with periodically decoding I-frames. Wang [7] proposes an algorithm to detect human face regions from dequantized DCT coefficients of MPEG video. This method is suitable for color

images with face regions greater than 48 by 48 pixels (3 by 3 MPEG macroblocks). Most of the gesture/activity detection related work use activity measurements from uncompressed images after a proper segmentation of human body parts. Major approaches for analyzing spatial and temporal patterns include Dynamic Time Warping (DTW) [8], Neural Networks (NNs), and Hidden Markov Models (HMMs) [9]. A method based on time-frequency analysis is proposed in [10] to detect periodic human motion with self-similar characteristics. The outline of the human body is used to detect the periodical relative limb movement in [11] by a template matching process. Principal component analysis method is one of the global approaches used by Yacoob and Black [12] for human activity recognition in uncompressed video sequences where motion measurements are used for segmented body parts.

In MIT Multimedia Lab. ([13], [14]) computer systems, that can follow people's actions, recognize their faces, gestures, and expressions, are developed. This technology enables to create "smart room" and "smart clothes" that can help people in day-to-day life without chaining them to keyboards, pointing devices or special goggles. Stillman [15] proposes a near real-time system where different type of sensors, such as microphones and cameras are spread throughout the environment. The system tracks the person in the environment, detects the face and recognizes the speech signal. Another system, that uses multimodal input (voice and machine vision), is implemented by Lucente [16]. The system uses single camera and processes approximately 10 frames per second with a latency of 0.2 seconds. Some simple gestures and voice commands are used to manipulate virtual objects. Siskind [17] presents a system, called Leonard, which classifies simple motion events, such as pick up and put down by using single camera input. Ayers and Shah [18] describe a system to recognize human actions by using the layout of the environment. They model the action recognition by a state machine and the system can generate key-frames for content-based video compression purposes. Haritaoglu et al [19] proposes a real time system (W4) for tracking people and their body parts in monochromatic imagery. It constructs dynamic models of people's movements to answer questions about what they are doing, and where and when they act.

Most of the activity/gesture detection techniques rely on the successful feature extraction. Our uncompressed domain activity/gesture recognition scheme uses model-based segmentation where high-level nodes are created from the combination of low-level segments via a feedback from graph matching part. We combine individual movements of the body parts by using the same bottom-up approach.

3 System Overview

Our proposed hierarchical system is given in Figure 1. The input of the system can be an uncompressed video from a Hi8 camera or a stored MPEG compressed video stream. First part of the system, posture and global activity recognition, uses DC-AC coefficients to recognize the posture from the outline of the human body. DCT frame differences are then used to recognize the overall movement of the recognized human sil-

houette. If needed, this information is used by the higher level part, activity/gesture recognition, to recognize gestures of the person from uncompressed video stream. First level, posture recognition part, is described in the following subsection.

3.1 Posture Recognition

Background elimination is performed by using the DC values for the current and background frames. Since the cameras are supposed to operate in a relatively fixed environment, we assume that background is known a priori and does not change over time. Hence, the block level process simply subtracts background from foreground image. Our proposed human detection algorithm basis on eigenspace representation of human silhouettes (1, Block A), using AC-DCT coefficients of frame blocks. To capture the intensity variations, first order AC coefficients (c_{01}, c_{10}, c_{11}) are used (Figure 3). DCT coefficient values capture the local directionality and coarseness of the spatial image. The vertical (horizontal) edges in uncompressed image correspond to high frequency component in the horizontal (vertical) frequencies and diagonal variations correspond to channel energies around the diagonal harmonics. Our approach is based on the observation that the structural information of human silhouettes can be captured from AC-DCT coefficients. In particular, the energy of blocks that is obtained by summing up the absolute amplitudes of the first order harmonics is used. To train our system, 200 pedestrian images for each posture type are used. Eigenspace matching block in Figure 1 is shown more detailed in Figure 2. The windowing and scaling blocks in our previous work [2] are replaced by a single scaling block due to the real-time implementation purposes. Since resizing and shifting a window throughout the test image is computationally very expensive, we calculate the width of the foreground object and choose the height of the window by using body proportions for standing man, standing man with one arm open, and standing man with two arms open. The window is then resized to compare the AC coefficients of the object with the coefficients of the training data set. The scaling operation is done in compressed domain [20].

Our goal is to find a compact representation of the human silhouette by computing the eigenvectors of the covariance matrix of the human body images. These eigenvectors represent a set of features which together characterize the variation between human images. The number of eigenvectors (M) is equal to the number of images in the training set. In our algorithm we use the best eigenvectors ($M' = 12$) with the highest eigenvalues. The following steps summarize the recognition (last block in Figure 2): a) Compute eigenvectors and eigenvalues from the training set of compressed human body images, b) given an input image, calculate a set of weights based on the input image and the M' eigenvectors by projecting the input image onto each of the eigenvectors, c) detect human regions by computing the distance between the mean adjusted input image and its projection onto human body space.

The training set of human images is $\Gamma_1, \Gamma_2, \dots, \Gamma_M$, and the average is $\Phi = (\Gamma_1 + \Gamma_2 + \dots + \Gamma_M)/M$. The difference of a human image from this average image is $\phi_i = \Gamma_i - \Phi$.



Figure 2. Eigenspace matching.

Our goal is to find a set of M orthonormal vectors, u_k and their eigenvalues β_k which best describes the distribution of the data by using the principal component analysis. u_k and β_k are the eigenvectors and eigenvalues, respectively, of the covariance matrix C :

$$C = \frac{1}{M} \sum_{n=1}^M \phi_n \phi_n^T = AA^T \quad (1)$$

where the matrix $A = \frac{[\phi_1 \phi_2 \dots \phi_M]}{\sqrt{M}}$. The matrix C is a N by N matrix and the calculation of eigenvectors and eigenvalues of this matrix is a difficult task. To reduce the computational complexity, the eigenvectors x_k and eigenvalues λ_k of the matrix $A^T A$ are computed. The eigenvectors u_k of matrix C can be computed as:

$$u_k = \frac{\sum_{l=1}^M \phi_l x_{kl}}{\sqrt{\lambda_k}} \quad (2)$$

and the eigenvalues are the same those matching x_k . Creating the vector of weights for an image is equivalent to projecting the image onto the human body space. The distance ϵ between the image and its projection onto the body space is the distance between the mean adjusted input image $\phi = \Gamma - \Phi$ and $\phi_f = \sum_{k=1}^{M'} \omega_k u_k$, its projection onto human body space, where $\omega_k = u_k^T (\Gamma - \Phi)$ for $k = 1, \dots, M'$.

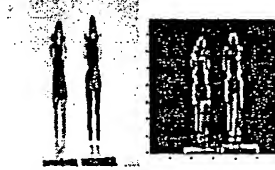


Figure 3. Left: Original Image, Right: AC-DCT values.

In our system, posture recognition result can help the system to decide between two possibilities, storing the frames with a particular posture, such as pointing to the right, in a database and/or processing input frames, starting with this particular posture, for activity recognition purposes for a certain number of frames or until detecting another particular posture. Next step is the global motion direction classification of the human body.

3.2 Motion Direction Estimation

DCT coefficients have been used by researchers to detect particular features. Peacock [21] used DCT image differences to estimate the motion direction by using 4x4 DCT blocks and 6 AC coefficients. However, 8x8 DCT blocks in MPEG sequences limit the performance of the feature detection technique used by Peacock and Pagliari. In this work (1, Block B), first we take the DCT difference of the current and previous frames and use the DC magnitude to eliminate stable areas without motion. We then compare three horizontal AC coefficient differences $C(0, 1)$, $C(0, 2)$, and $C(0, 3)$ with the ver-

tical AC coefficient differences $C(1, 0)$, $C(2, 0)$, and $C(3, 0)$, respectively. With this method, we can classify the up/down, left/right, and diagonal activities of the human body silhouette in real-time. Uncompressed domain gesture/activity recognition algorithm is given in the next sub-section.

3.3 Gesture/Activity Recognition

The uncompressed domain gesture/activity recognition part consists of two subparts, low and high-level processing (1, Block C). The low-level part performs human detection and extracts parameters for the abstract graph representation of the image being processed. The high level part uses HMM algorithm to determine the movements of the person's body parts, and uses a distance classifier to detect specific gestures. Figure 4 displays a sample frame processed by the algorithm.

3.3.1 Low-level processing

Region extraction: Region extraction block performs skin area detection. The output of this block is a frame size buffer and is fed into the next algorithm block. Skin area detection identifies skin regions in the input image by comparing color values of each pixel to a human skin color model. Considering the processing time limit of a real application, we use YUV color model.

Contour extraction: Contour following uses a 3x3 filter to extract the boundary of each object region obtained by previous block.

Ellipse fitting: In this step, the algorithm finds the ellipse parameters which can optimally describe the boundaries extracted. Ellipses with shape preserving deformations help to overcome the occlusion problem because of different body parts, such as torso occlusion due to hand.

Object modeling by invariant shape attributes: The descriptors of body parts are classified into two groups: unary (compactness, eccentricity) and binary (ratio of areas, relative position and orientation, adjacency) attributes.

Graph matching: Each extracted region modeled with ellipses correspond to a node in the graphical representation of human body. Face detection allows to start initial branches efficiently and reduces the complexity. Each body part and meaningful combinations represent a class (ω) where the combination of binary and unary features are represented by a feature vector (X) that is computed off-line. For the purpose of determining the class of the feature vectors a piecewise quadratic Bayesian classifier with discriminant function $g(X)$ is used. The generality of the reference model attributes allows the detection of different postures while the conditional rule generation (r) decreases the rate of false alarms.

3.3.2 High-level processing

The high-level algorithm part consists of two subparts. In the first part, the position of each body part, identified by the low-level algorithm, is tracked. HMM algorithm is applied to identify the movement of those parts. In our application, we assume that each body part has its own freedom of motion and the activity recognition for each part is achieved by using several HMMs in parallel. Combining the outputs of the HMMs

to generate scenarios is an application dependent issue. The movement of the body parts are described as a spatio-temporal sequence of feature vectors that consist of the direction of the body part movement. Since we use discrete HMMs, we generate 8 directional codewords. We check the up, down, right, left, and circular movements of the body parts. Our models are trained using Baum-Welch algorithm. Note that the detected movement of the body part may be a part of a more complex activity. We check the current pattern and combine it with the immediately following one and generate a new pattern. Using dynamic programming we calculate the probabilities for the first and combined patterns and choose the pattern with the highest probability as the recognized activity. If the probability of the observed activity is below a threshold, we don't classify the activity. Furthermore, we use the gap between different gestures/activities, e.g., moving the hand out of camera, stopping the body for a while. Another feature in the activity recognition is the speed of the body parts. We use the speed of each body part (slow/fast) for one activity period as an additional input for the classification. Next step is the generation of a feature vector by using the observed activities of the body parts. The activity feature vector is compared with the known activities via a distance classifier, based on the Mahalanobis metric. The quadratic Mahalanobis distance classifier is employed to combine these movements together to detect the specific gesture made by the person in the scene. While the low-level processing is executed on frame basis, high-level processing takes several consecutive frames to identify the movements.

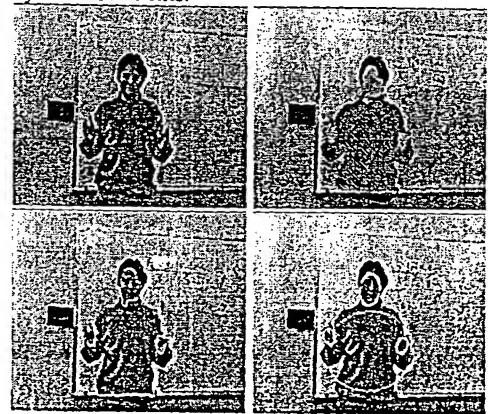


Figure 4. Snapshot of a sequence for each low-level algorithm block: Original frame, skin color detection, contour following, ellipse fitting.

3.4 Testbed Architecture

In our prototype system, a single camera node is composed of a standard camera and a TriMedia video processing board. A TriMedia board has one TM1300 TriMedia processor, which is specialized for media processing that allows Windows and Macintosh platforms to take advantage of the TriMedia Processor via PCI interface. A 32-bit TM1300 processor has its own dedicated memory and a five issue VLIW (Very Long Instruction Word) CPU together with several co-processors. After initialization, the Video In unit automati-

cally stores the input image into an assigned buffer. When the loading of one frame is finished, an interrupt is arisen to notify the CPU. The CPU can then copy the data in the buffer and supply empty buffers for new frames. Video Out unit puts the frames in the supplied buffers to an output video stream when a frame is processed, the CPU is notified by an interrupt. In our system, TM1300 processor runs at 100MHz, providing a peak performance of 500 MOPS.

4 Results

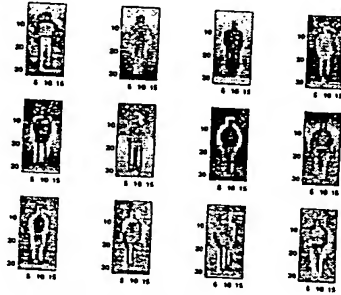


Figure 5. 12 eigenimages for standing man (upsampled).

In this section, we present results for different steps of the algorithm for performance evaluation. We use two TriMedia video capture boards, that can communicate via shared memory. Compressed and uncompressed domain algorithms are tested on separate boards each connected to a video camera. When a human body activity is detected in the compressed domain, the activity/gesture recognition algorithm starts running on the second board for a more detailed analysis in the uncompressed domain. We can achieve a processing rate of 25 frames per second for posture and activity/gesture recognition algorithm blocks in both domains.

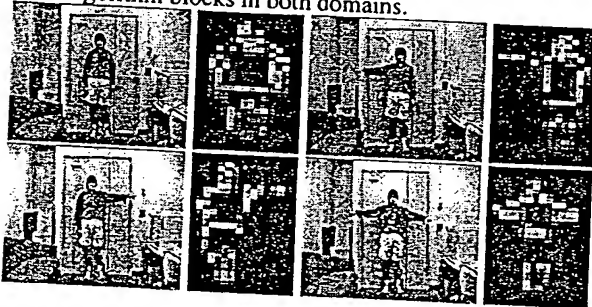


Figure 6. Four correctly classified postures and corresponding AC coefficients (standing, right arm open, left arm open, both arms open)

4.1 Compressed Domain

Figure 6 displays four different postures with corresponding AC images. 12 eigenvalue images for "standing man" are displayed in Figure 5 to show how a compact representation of human silhouette is obtained by computing the principal components of the energy distribution of human bodies.

Figures 7 displays vertical and horizontal motion detection respectively. Figure 8 displays the first key frame where posture (left arm open) is detected via eigenspace matching of AC coefficients. The subsequent frames are processed with reference to this frame where DCT coefficients differences between frames are computed to detect motion direction. By

using human silhouette detected in the key frame and human body proportion, left arm motion is detected. Note that any activity detected in compressed domain can be used as a trigger for more detailed analysis in uncompressed domain.

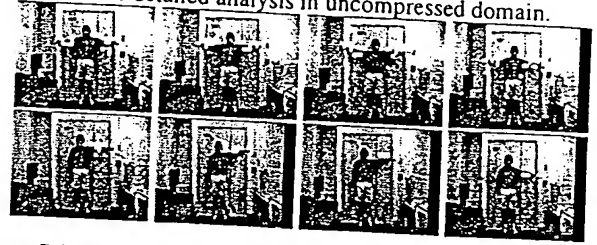


Figure 7. Vertical motion detected frames (White blocks correspond to DCT blocks with vertical motion): Arms Up/Down

The algorithm performance is tested for real-time video inputs. DCT coefficients are calculated for each frame and different human postures are recognized by using our proposed eigenspace matching algorithm. 78 open arm posture (left and/or right arm open) out of 83 sequences, and 58 standing man posture out of 60 sequences are correctly classified.



Figure 8. First and second figures: Detected key frame and AC coefficients (left arm open), Figures 3-6: Activity detected frames.

4.2 Uncompressed Domain

86% of the body parts (head, torso, hand1, and hand2) in the processed frames are correctly classified, the rest is the miss and false classification.

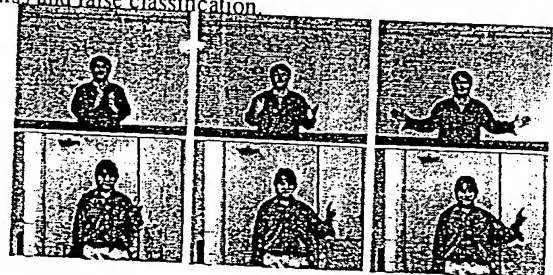


Figure 9. Example frames: Top: Opening and closing arms, Bottom: Pointing left

Some of the activities are opening and closing arms, pointing left and right, walking to left/right, moving up and down, circling hands (clockwise/counter-clockwise), etc. (Figure 9). Note that for each activity pattern of the body parts we use 30 training sequences as the input of the Baum-Welch algorithm. The process of constructing complex patterns by combining the sub-patterns of the body parts is achieved by the distance classifier [22]. Activity recognition results for some of the

Pattern	Correct	False
Pointing Right	18	3
Pointing Left	18	1
Open Hands	13	2
Close Hands	13	2
Moving Up	16	0
Moving Down	16	0

Table 1. Results for activity classification.

example patterns are given in Table 1. Most of the false detections of the exemplar activities occur because of the classification errors of the body parts. Note that, smaller body parts, such as hands and fingers, have higher false classification ratios than the larger ones.

We observe that different activity patterns can have overlapping periods (same or similar patterns for a period) for some body parts. Hence the detection of start and end times of activities is crucial. As stated earlier, to detect the start and end time of a gesture, we use the gap between different gestures/activities. We also check the current pattern and combine it with the immediately following one and generate a new pattern. Furthermore, the speed of the body parts is used as another feature in the activity recognition. Note that the combination of the activities of the body parts make this detection more reliable. For example, consider a person who, after pointing toward an object (head and torso are idle while hand points out), starts walking to the door (all body parts move toward door, and hands are moved down). Although body parts have different importance in activity patterns, all the body parts' activities contribute with different weights to the detection of the start and end times of the overall activity pattern. Figure 10 displays fitted ellipses for some of the activity patterns, namely pointing one hand, opening and closing hands, and left-right movement of the body.

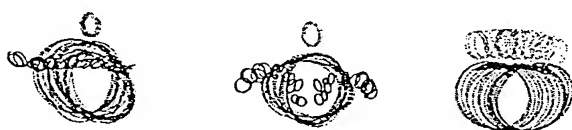


Figure 10. Cumulative motion (fitted ellipses) for different activity patterns: Pointing, opening and closing arms, left-right movement.

5 Conclusions

This paper presents a real-time system for posture recognition and activity detection in the compressed domain. The major contribution of the overall algorithm is to connect available data in low-resolution compressed domain to high level semantics. We show that with this technique we can reduce computational complexity, avoid dependency on correct segmentation, and reduce storage area and bandwidth requirements. Furthermore, we show the interaction of this system with our activity recognition system in the uncompressed domain in order to recognize specific gestures for different type of applications. Our compressed domain technique is simple to implement and yet powerful, which is suitable for surveil-

lance systems with remote monitoring, automatic alarm and storage capabilities that require efficient compression techniques.

References

- [1] B. Zalud, "2002 Industry Forecast Study, Security Yin-Yang: Terror Push, Recession Drag", Security Magazine, 2002
- [2] "Human Detection in Compressed Domain", IEEE International Conference on Image Processing (ICIP), Greece, 2001
- [3] "A Hierarchical Human Detection System in (Un)compressed Domains", To appear in IEEE Transactions on Multimedia, Vol. 4, No. 2, 2002.
- [4] A. Pentland, R. Picard, and S. Sclaroff, "Photobook: Content-based Manipulation of Image Databases", International Journal of Computer Vision, Vol. 18, No. 3, pp. 233-254, 1996.
- [5] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna and T. Poggio, "Pedestrian Detection Using Wavelet Templates," Proc. of CVPR, pp. 193-199, June 1997.
- [6] D. Schonfeld and D. Lelescu, "VORTEX: Video retrieval and tracking from compressed multimedia databases - template matching from MPEG2 video compressed standard", Proc. of SPIE Conference on Multimedia and Archiving Systems III, Vol. 3527, pp. 233-244, 1998.
- [7] H. Wang and Shih-Fu Chang, "A Highly Efficient System for Automatic Face Region Detection in MPEG Video Sequences", IEEE Trans. on Circuits and Systems for Video Technology, Vol. 7, No. 4, pp. 615-628, Aug. 1997.
- [8] K. Takahashi, S. Seki, R. Oka, "Spotting Recognition of Human Gestures from Motion Images," T.R. IE92-134, 1992.
- [9] M. Brand, N. Oliver, and A. Pentland, "Coupled Hidden Markov Models for Complex Action Recognition," IEEE CVPR, San Juan, 1997.
- [10] R. Cutler and L. Davis, "Real-Time Periodic Motion Detection, Analysis and Applications", CVPR, pp. 326-332, 1999.
- [11] C. Curio, J. Edelbrunner, T. Kalinke, C. Tzomakas, W. von Seelen, "Walking Pedestrian Recognition", International Conf. on Intelligent Transportation Sys., pp. 292-297, 1999.
- [12] Y. Yacoob and M. J. Black, "Parameterized Modeling and Recognition of Activities", ICCV, pp. 120-127, 1998.
- [13] A. D. Wilson, A. F. Bobick, "Realtime Online Adaptive Gesture Recognition," International Conference on Pattern Recognition, pp. 270-275, 2000.
- [14] A. Pentland, "Smart Rooms, Smart Clothes," International Conference on Pattern Recognition, pp. 949-953, 1998.
- [15] S. Stillman and I. Essa, "Towards Reliable Multimodal Sensing in Aware Environments," Workshop on Perceptive User Interfaces, Orlando, November 2001.
- [16] M. Lucente, G.-J. Zwart, and A. D. George, "Visualization Space: A Testbed for Deviceless Multimodal User Interface," Computer Graphics, Volume 31, Number 2, May 1997.
- [17] J. M. Siskind, "Visual Event Classification via Force Dynamics," AAAI 2000, pp. 149-155.
- [18] D. Ayers and M. Shah, "Monitoring Human Behavior from Video Taken in an Office Environment," Image and Vision Computing, Volume 19, Issue 12, 1 October 2001, pp. 833-846.
- [19] I. Haritaoglu, D. Harwood, and L. Davis, "W4: A Real Time System for Detecting and Tracking People," Third Face and Gesture Recognition Conference, pp. 222-227, 1998.
- [20] S. F. Chang and D. G. Messerschmitt, "Manipulation and Compositing of MC-DCT Compressed Video," IEEE Journal on Selected Areas in Communications, vol. 13, no. 1, pp. 1-11, Jan. 1995.
- [21] A. M. Peacock, D. Renshaw, and J. Hannah, "Motion Direction Estimates from Differenced DCT Images," Electronics Letters, Vol. 37, No. 3, pp. 163-164, 2001.
- [22] "A Bottom-Up Approach for Activity Recognition in Smart Rooms", To appear in IEEE ICME Conference, August 2002.

Parallel Architecture for Video Processing in a Smart Camera System

Tiehan Lv, Burak Ozer, Wayne Wolf

Department of Electrical Engineering
Princeton University, Princeton 08544
{lv, iozer, wolf}@ee.princeton.edu

Abstract

In this paper, we present our research on parallel architectures for a smart camera system. We analyze the available data independencies for a particular application, namely human detection and activity recognition, and discuss the potential architectures to exploit the parallelism resulted from these independencies. Three architectures—VLIW, symmetric parallel architecture and macro-pipeline architecture are discussed and their performances are presented.

1. Introduction

Recent developments in sensor technology enable to build smart cameras that can operate stand-alone in real-time. Smart cameras can be used to detect people and recognize their activities in an application environment, such as room, plane, car, or security checkpoint. Furthermore, the results of smart camera analysis can be used to control the operation of devices in these application environments. Our proposed smart camera algorithm is divided into two major blocks, low level and high level processing parts. Low level processing block covers skin color extraction, background elimination, region extraction, fitting ellipses to the body parts, and graph matching to detect the presence of a person in the environment. High-level part consists of Hidden Markov Model (HMM) algorithm blocks combined with a squared Mahalanobis distance classifier to recognize different activities of a person. The details of the overall algorithm can be found in [5].

Video processing for smart camera requires intensive computation, so it may be necessary to employ multiple processors for enough processing capability. In such a case, the organization of the processors rises as an important issue. In order to distinguish the organization of multiple processors from the architecture of a single processor, we call it the macro-architecture.

In this paper, we analyze the data independency for our human detection and activity recognition algorithm and how three different architectures, namely VLIW, macro-pipeline, and symmetric architectures can be used to exploit these independencies.

The contributions of this paper are as follows:

- 1) Identify the major data independencies of a smart camera application, which can represent a category of video processing program.
- 2) Analyze the relationship between data independencies and the parallelism, which can be exploited by hardware.
- 3) Compare the performance and other trade-offs of different parallel architectures

Besides the algorithm development, hardware design is one of the most important issues for a real time system. J. A. Watlington and V. M. Bove propose a data-flow model for parallel media processing [1]. L. Davis et al. develop a multi-perspective video system in University of Maryland [4]. Jason Fritts et al. evaluate the characteristics of multimedia applications for media processors [7]. Researchers also pay attention to multiprocessor architecture. Simultaneous multi-threading is proposed by Tullsen et al [8]. Hammond et al propose to use single chip multiprocessor architecture. An IMAGINE processor is being developed at Stanford University, which has explicit programmable communication structure [9]. Great effort has been devoted to develop real time human tracking systems. One of them is Pfindex [11]. Pfindex is developed at MIT media lab where the system uses Maximum A Posteriori Probability (MAP) approach to detect and track people by using 2D models. W4 is another real time human tracking system [10], where the background information should be collected before the system can track foreground objects. A more detailed introduction of work concerning human tracking is given by Pentland [3].

The rest of the paper is organized as follows. Section 2 describes the current smart camera system. Section 3 discusses different data independencies and parallelisms. Section 4 discusses VLIW architecture and instruction level parallelism, inter-frame data independency and symmetric architecture, and inter-stage independency and macro-pipeline architecture. Section 5 concludes the paper.

2. Hardware and Software Architecture

The smart camera system is based on a PC platform. Two TriMedia boards are installed to PCI slots of a PC. Figure 1 shows this hardware environment.

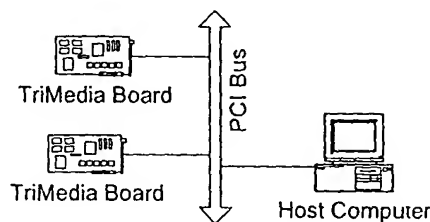


Figure 1: Hardware Environment

A TriMedia board has a specialized media processor, namely a TM1300 TriMedia processor. The TriMedia board is designed to cooperate with PCs or Macintosh computers via PCI interface. Multiple TriMedia boards in a host can communicate with each other via shared memory. A TM1300 processor is a 32-bit five issue VLIW (Very Long Instruction Word) processor with several supporting coprocessor units such as a video input and a video output units providing a convenient interface for video I/O stream. Each TriMedia processor has its own dedicated memory.

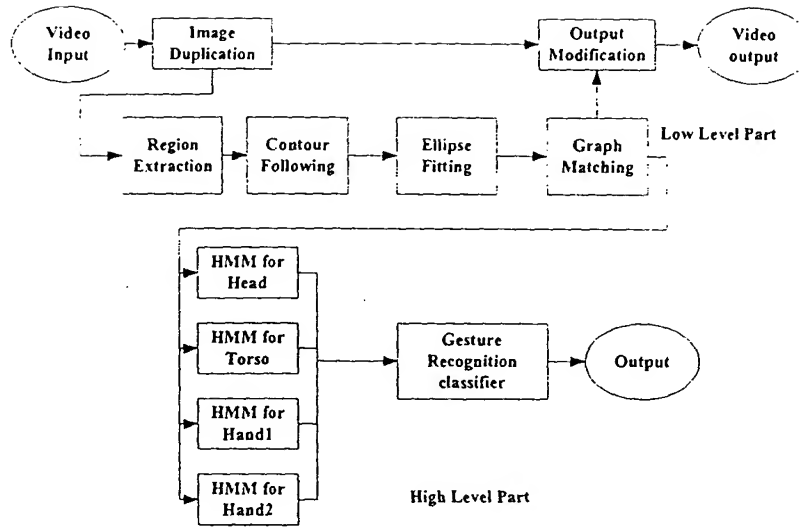


Figure 2: Software Architecture

Our human detection and activity recognition algorithm consists of two parts, low and high-level processing. The low-level part performs human detection and extracts parameters for the abstract graph representation of the image being processed. It consists of region extraction, contour following, ellipse fitting and graph matching algorithm stages. Region extraction performs background elimination and skin color detection to identify foreground objects and skin regions. The contour following stage then extracts the boundary information of these regions. Ellipse fitting is used to find a set of abstract parameters describing these regions based on the boundary information provided by contour following. Graph matching thus processes these abstract descriptors to identify human body parts. The high level part uses HMM algorithm to determine movements of the person's body parts, and uses a distance classifier to detect specific gestures. A higher level of HMM algorithm block is being integrated into the system to provide the system ability to recognize more complex gestures. Figure 2 shows the current software architecture for implementing the algorithm on one TriMedia board. Some snapshots of the sequence for low-level algorithm blocks are displayed in Figure 3. 86% of the body parts in the processed frames are correctly classified; the rest is the miss and false classification. Some of the example frames from the recognized activity sequence (opening/closing arms) are displayed in Figure 4. Figure 5 displays cumulative motion of the body parts. Some of the activities are opening and closing arms, pointing camera and door, walking to the door and camera, moving up and down, circling hands (clockwise/counter-clockwise), etc.



Figure 3 Snapshot of a sequence for each algorithm block
Left: Skin color detection. Middle: Contour following. Right: Superellipse fitting



Figure 4 Example frames from the recognized activity sequence: Opening and closing arms



Figure 5 Cumulative motion of body parts (fitted super-ellipses) for opening and closing arm activity

3. Parallelism and parallel architecture

The input frame size for the smart camera is 384x240 pixels. For each pixel, three bytes are used to represent Y, U, V components. The input frame rate is 30 frames per second. Therefore, the input data rate is about 8Mbytes/sec. The high input rate posts demanding request on the processing unit. It must have the processing ability to finish the processing in real time. To achieve this processing ability, parallelism of the input data and of the algorithm needs to be exploited.

Parallelism is always associated with data and resource independencies. If two processing units do not share writable data and exclusive accessible resources, they can perform processing simultaneously. First, we can focus on data independency under the assumption that we can always offer enough processing elements. For our smart camera application, there are three different kinds of data independencies. The first one is intra-frame data independency. For low-level pixel based processing, such as background elimination, only the information about a pixel is needed for processing, so the whole frame can be divided into several independent parts. A weaker form of intra-frame data independency also exists. If the information about the neighboring pixels is also required, a frame can be divided to several parts with small amount of overlapping to be processed independently. After each part is processed, additional work needs to be done to merge the results together.

The second kind of data independency is the inter-frame independency. For image segmentation and abstraction, the information about the relationship between frames is not required, so each frame can be processed independently. If the intra-frame data independency is based on special partition, the inter-frame data independency will be based on temporal partitioning.

The third one is the independency of the sequential algorithm components. The overall recognition algorithm consists of several sequential processing stages. Therefore, a precedent processing stage can start processing the new data before its successor finishes the processing.

If we allocate several functional units into one processor, we can explore the data independency that can be converted to instruction level parallelism. We can also form the simultaneous multi-threading structure to explore the data independency that can be converted to thread level parallelism or process level parallelism. A third option is to use multiple processors to explore the data independency that can be converted to process level parallelism.

4. Parallelism and architecture

1) Instruction level parallelism & VLIW architecture

Since the input frame size is not very large, the intra-frame data independency is converted to fine-grained parallelism. This parallelism does not have enough granularity to form thread level or process level parallelisms, so we convert this parallelism to instruction level parallelism, which can be explored by VLIW or superscalar architectures. The instruction level parallelism resulted from the intra-frame data independency can be explicitly expressed in an executable file. This means that instruction level parallelism is available during the compile-time. Both VLIW and superscalar processors can exploit static instruction level parallelism. While a superscalar processor is more efficient for processing dynamic instruction level parallelism, a VLIW processor requires less hardware effort in trade of less efficient when dealing with instruction level parallelism only available at runtime. In this work, since we want to explore static instruction level parallelism, VLIW architecture processors are used.

To exploit instruction level parallelism resulted by the intra-frame data independency, we perform manual optimization of the program for a VLIW processor—TriMedia 5-issue VLIW processor, TM1300. Table 1 shows the major characteristics of a TriMedia TM1300 processor.

	Units	#Units	Latency (Cycles)
Functional Units	Constant	5	1
	Integer ALU	5	1
	Load/Store	2	3
	DSP ALU	2	2
	DSPMUL	2	3
	Shifter	2	1
	Branch	3	3
	Int/Float MUL	2	3
	Float ALU	2	3
	Float Compare	1	1
	Float sqrt/div	1	17
#Register		128	
Instruction cache		32KB, 8 way	
Data cache		16KB, 8 way	
#Operation slots/instruction		5	

Table 1: Features of TriMedia processors

The first step is to use loop fusion and loop unrolling to increase the basic block size and thus increase available parallelism. Loop fusion merges several loops together to increase the size of a loop body. Loop unrolling put more work into a loop to reduce trip count. For our processing algorithm, the loop unrolling can be very effective, since the large loop level parallelism is available. In addition, the overhead of loop unrolling in start-up and close-up code can be minimized since the size of the loop is usually known at the compile-time. Although loop unrolling and loop fusion can increase the code size for each trip, we find out that the basic block sizes do not increase significantly. The reason is that for processing each pixel, the calculation of absolute value and the determination a variable is within a specific range is needed. Those calculations are converted to branches, which limit the size of the basic blocks and thus limit the parallelism that can be exploited by a TriMedia VLIW processor. TriMedia processors offer a solution to this problem, special conditional instructions. For example, the instruction "iabs" can calculate the absolute value of the input data without help of branch instructions. After the optimization of the program by using these special instructions, the instruction level parallelism is increased, and the running times of the background elimination and skin color detection are reduced significantly.

Another optimization technique is reducing branches by convert control flow dependency to data dependency. In our algorithm, contour following, the instruction level parallelism is limited by the control flow dependency. The critical control flow structure is displayed on the left-hand side of Figure 6. Although if-conversion can remove some branches, for such a control flow dependency, if-conversion may fail. To increase the available parallelism, we convert the control dependency to data dependency by evaluating the condition in each if statement. The results are put together into one single byte where each bit represents the result from each evaluation. A result table is built into the algorithm, so we can get the result by using the byte as the index of the table. The control flow after conversion is displayed on the right-hand side of Figure 6. Note that by using this method, the branches are eliminated, and the instruction level parallelism in contour following algorithm block is increased.

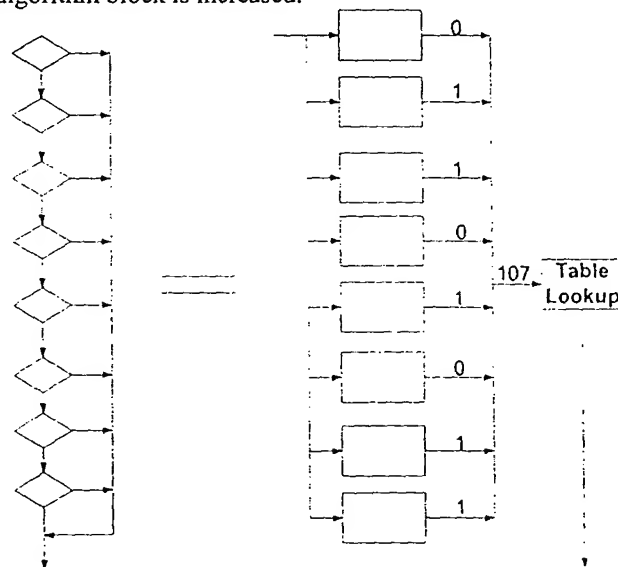


Figure 6: Branch reduction

By using manual optimization, we increase the available instruction level parallelism and thus improve the performance of the smart camera application. The results are displayed in Figure 7 and Figure 8.

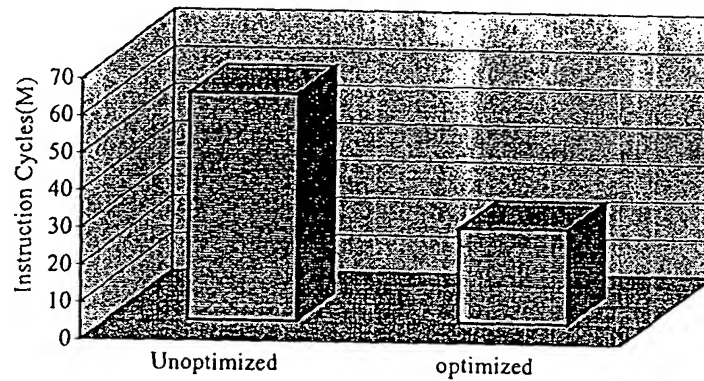


Figure 7: Instruction Cycles for process ten frames

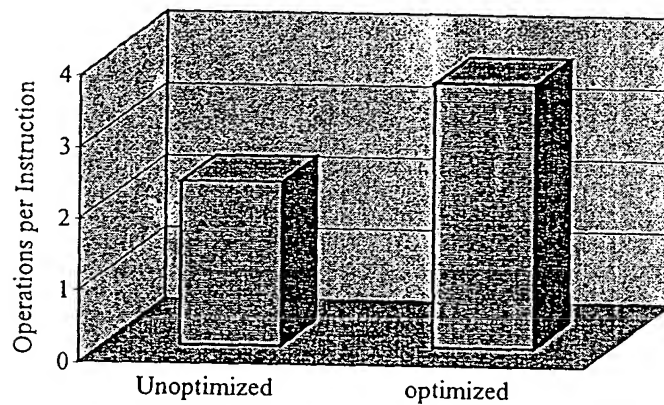


Figure 8: Available Parallelism

The results show that although the amount of parallelism resulted by intra-frame data independency is large, the instruction level parallelism is limited where instruction level parallelism is limited by control dependency. By using special conditional executed instruction, some control dependencies can be eliminated, but those control dependencies that cannot be eliminated will limit the instruction level parallelism.

Another limit of instruction level parallelism is that it is a fine-grained parallelism. It is hard to convert those coarse-grained data independencies such as inter-frame independency into instruction level parallelism.

From architectural point of view, instruction level parallelism also meets some limitations. Increasing global interconnection delay will prevent processor designers to build large amount of functional units into one single processor. Increasing issue width of a processor will also increase

the number of ports of the register files inside the processor. As a result, this will increase the complexity of the register file organization and increase the access time of the register file.

Starting from next section, we will evaluate alternative architectures, which can exploit thread level and process level parallelisms.

2) Inter-frame Level Parallelism & Symmetric Architecture

Another level of data independency available for our smart camera application is inter-frame data independency. This is a coarse-grained data independency. It can be converted into thread level or process level data independency. Both SMT (Simultaneous Multithreading) and CMP (single chip multi-processor) architectures can exploit process level parallelism. The SMT architecture is not suitable to our current implementation. The reason is that SMT architecture is more suitable for complimentary processes that can share functional units efficiently. However, to take advantage of inter-frame data independency, the processes are identical to each other. In this section, we use the CMP architecture to exploit inter-frame level parallelism. The proposed architecture is shown in Figure 9. A processor pool is employed to process the input frames. The output buffer is used to keep the sequential information among frames. Processing time distribution among algorithm blocks is shown in Table 2.

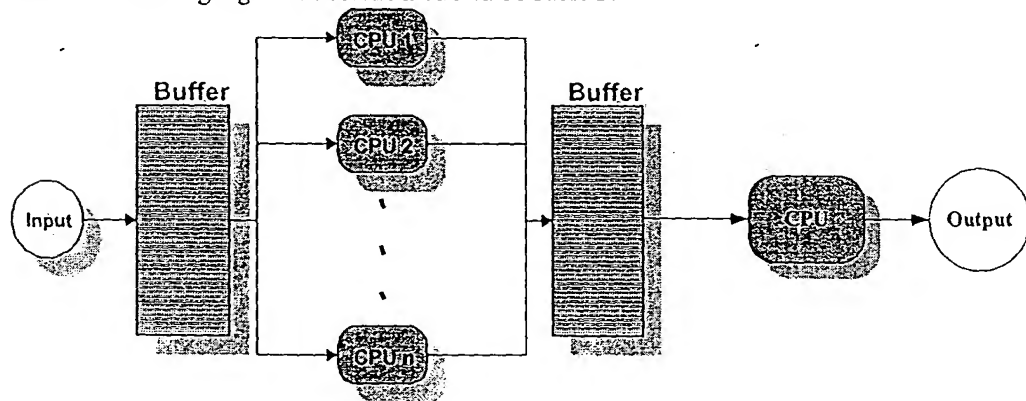


Figure 9: Symmetric Parallel Architecture

	Region	Contour	Ellipse	Match	High Level	Total
Percentage	44.09%	51.99%	0.60%	3.23%	0.09%	100%

Table 2: Processing time distribution among algorithm blocks

Under the assumption that there is enough input data for processing, the throughput of the symmetric parallel architecture is displayed in Figure 10, where the throughput is scaled to the throughput of a single processor. Series1 is the performance under the assumption that communication cost is negligible. Series 2 is the performance where the communication cost is 20% of the average processing time of a frame. The performance is calculated according to the following formula.

25 of 41

$$ProcessingTime = \left(\frac{LowLevel ProcessingTime}{\#Processors - 1} \right) + HighLevelProcessingTime + CommunicationCost$$

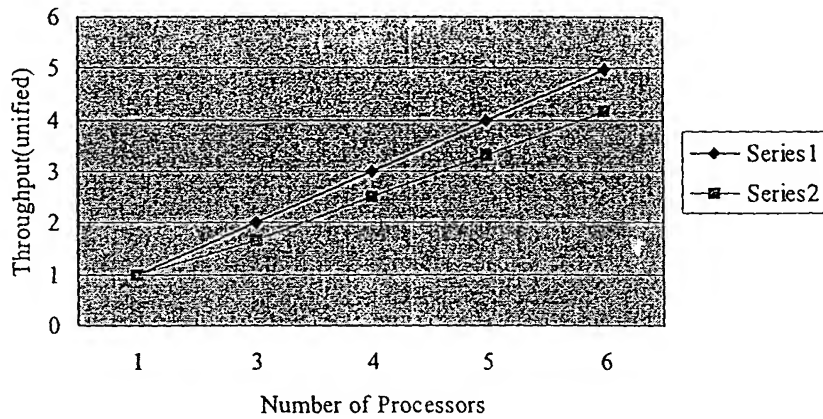


Figure 10: Performance of symmetric architecture

3) Inter-stage level Parallelism & Pipeline Architecture

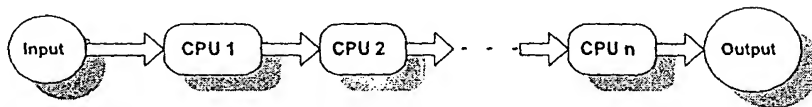


Figure 11: Macro-pipeline Architecture

The previous discussion is related to the data independency inside the input data. The inter-stage level parallelism is resulted from the data flow structure. The data flow structure can be mapped to pipeline structure, which is also a CMP architecture.

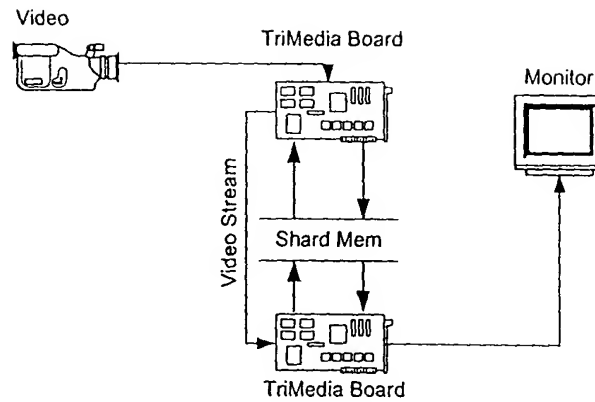


Figure 12 Pipelined System

A pipelined smart camera is built. The system consists two TriMedia boards, which can communicate with each other via a shared memory. Figure 12 shows the structure of the pipelined system. The software structure and the allocation of the algorithm in processes are shown in Figure 13. The two processes are assigned to two TriMedia boards.

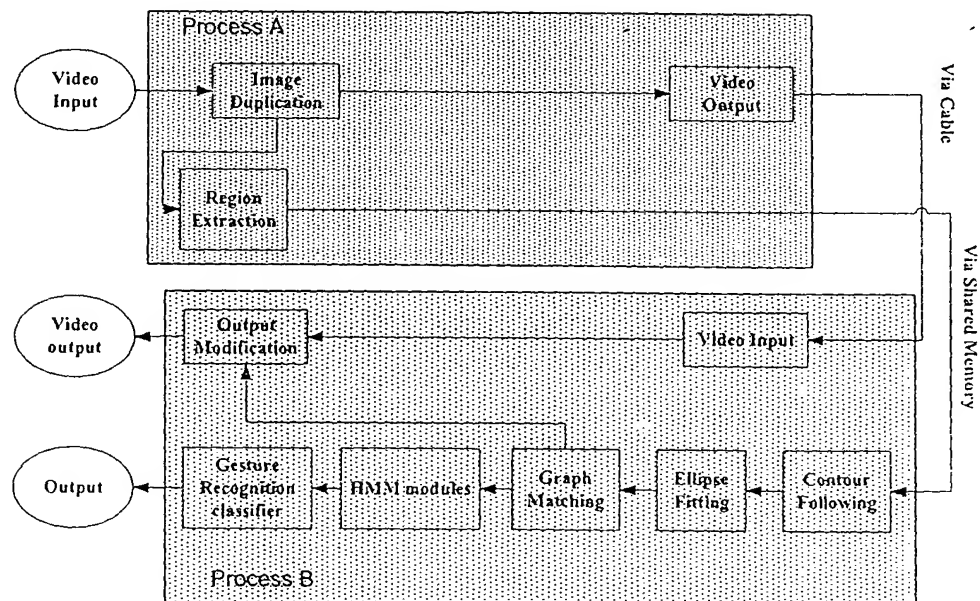


Figure 13 Software Structure in Pipelined System

Figure 14 displays the performance of pipelined architecture. Series 1 shows the throughput when communication costs are zero. Series 2 shows the throughput when communication costs are 20% of the total processing time. The performance improvement is limited the fact that contour algorithm block needs 52% of the processing time and it has to be located in one processor. The processing time is calculated by the following formula.

$$ProcessingTime = Max(ProcessingTimeInProcesses) + CommunicationCost$$

A benefit for this approach is that the CPUs can have different capacity. For example, if a CPU is only required to process region extraction or contour following, it does not need to have floating point unit.

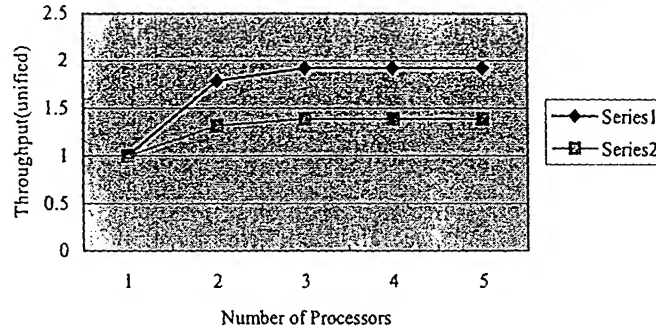


Figure 14: Throughput of pipeline architecture

4) Comparison between architectures and data independency

Independency	Architecture	Dedicated architecture	Performance
Intra-frame independency	VLIW/Superscalar	VLIW—TriMedia 1300 Processor	3.7 x
Inter-frame independency	CMP/SMT	Symmetric parallel architecture	5 x ^a
Inter-stage independency	CMP/SMT	Macro-pipeline architecture	1.3 x ^b

Table 3: Performance Comparison

The performances given in Table 3 are scaled to a single processor with issue width one. From the results, the symmetric parallel architecture can provide best performance, which is five times faster than a single processor. However, to achieve such a performance upgrade, an infinite number of processors are needed. When the number of processors is limited, the performance upgrade is limited as shown in Figure 10. In addition, different architectures have different advantages and disadvantages. AVLIW architecture can exploit fine-grained parallelism but is limited by global bus delay and complexity of register files. A symmetric parallel architecture can provide almost linear speedup but it is limited by the fact that the algorithm should spend most of its time on intra-frame processing. A macro-pipeline structure can trim hardware to the need of each algorithm stage but its performance is limited by unbalanced processing time distribution among algorithm stages. A better way to exploit the available data independencies is to combine these architectures together to provide better performance.

^a Under assumption that there is a 20% communication cost.

^b Under assumption that there is a 20% communication cost. The processing time distribution among algorithm blocks is shown in Table 2.

5. Conclusions and Future Work

This paper discusses the parallel architectures for a smart camera system. We analyze the available data independencies inside the application and the potential architectures to exploit the parallelism resulted from these independencies. Three architectures—VLIW, symmetric parallel architecture and macro-pipeline architecture are discussed and their performances are presented. A potential optimum architecture is a heterogeneous architecture by combining these architectures together. Our future work will investigate the issue about heterogeneous architectures.

References

- [1] J.A. Watlington and V.M. Bove, Jr., "A System for Parallel Media Processing," *Parallel Computing*, Dec 1997
- [2] A. Pentland, "Looking at People: Sensing for Ubiquitous and Wearable Computing", *IEEE PAMI*, Vol 22, No 1, pp. 107-119, Jan. 2000
- [3] L.S. Davis, E. Borovikov, R. Cutler, and T. Horprasert, "Multi-perspective Analysis of Human Action", *Third International Workshop on Cooperative Distributed Vision*, 1999
- [4] I.B. Ozer, W. Wolf, A.N. Akansu, "Relational Graph Matching for Human Detection and Posture Recognition", *SPIE, Photonic East 2000, Internet Multimedia Management Systems*, Boston, MA November 2000
- [5] I.B. Ozer, and W. Wolf, "Video Analysis for Smart Rooms", *SPIE, ITCOM 2001*, Denver, CO, Aug 2001
- [6] J. Fritts, W. Wolf, and B. Liu, "Understanding multimedia application characteristics for designing programmable media processors", *SPIE Photonics West, Media Processors '99*, San Jose, CA, pages 2-13, Jan 1999
- [7] D.M. Tullsen, S. J. Eggers, H.M. Levy, "Simultaneous Multithreading: A Platform for Next-Generation Processors", *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995, pages 392-403
- [8] B. Khailany, W.J. Dally, S. Rixner, U.J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, and A. Chang. "Imagine: Media Processing with Streams," In *IEEE Micro*, Mar/April 2001
- [9] L. Hammond, B. Nayfeh, and K. Olukotun. A single-chip multiprocessor. *IEEE Computer*, 30(9), pp.79--85, September 1997
- [10] Ismail Haritaoglu, David Harwood and Larry S. Davis, "A real Time system for Detecting and Tracking People", *3rd international conference on face and gesture recognition*, Mar 1998, Nara, Japan
- [11] Wren, C.R., Azarbayejani, A., Darrell, T., Pentland, A.P., "Pfunder: Real-time Tracking of the Human Body", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19:7, July 1997

Smart Camera System Design

Tiehan Lv, Burak Ozer, Wayne Wolf
Department of Electrical Engineering,
Princeton University, Princeton
NJ 08544, USA

e-mail: {lv,iozer,wolf}@ee.princeton.edu

Abstract

This paper describes a smart camera system developed at Princeton University. The system is used as a real time visual surveillance system that can detect the presence of a person and recognize his/her activities in an indoor environment. The paper outlines the structure of the current prototype system that uses multiple cameras, each with its own video signal processor (VSP), and which is able to recognize several different gestures in a fixed background at a speed of 20 frames/second. Furthermore, this paper discusses possible structures and tradeoffs of the system.

1 Introduction

This paper describes a smart camera project conducted at Princeton University. The purpose of this project is to develop a real time visual surveillance system which is able to detect the presence of human beings in an indoor environment and recognize their activities. A prototype system has been built to detect a single person and recognize his/her gestures in a fixed environment. The system is able to process about 20 frames per second. Our current system has two smart camera nodes hosted by a PC. Skin color extraction, background elimination, region extraction, fitting ellipses to the body parts, and graph matching are used to identify the presence of a person. An abstract model is extracted to describe the body parts where the parameters of the model are then fed into parallel HMM algorithm blocks to track and classify the movement of the parts. Finally, a high-level classifier combined with the HMM algorithm determines specific gestures of the person.

Background elimination simplifies the procedure to find the foreground objects, but this approach requires a-priori knowledge about the background information. L. Davis et al. give an elaborated discussion of using background elimination to identify the foreground objects [4]. A fuzzy pattern matching method for face detection is proposed by H. Wu et al. [8], where Farnsworth color system is employed for skin color recognition. F. Solina and R. Bajcsy use superquadrics for shape representation [9]. This representation is used in our smart camera system to describe the human body parts of the person in the scene. Several different approaches have been developed for human activity recognition such as Hidden Markov Model, Dynamic Time Warping, and Dynamic Bayesian Networks [10],[11],[12],[13]. In our algorithm, we use HMM model combined with Mahalanobis distance classifier to determine the gestures.

Hardware design is important for the development of a real time system. J. A. Watlington and V. M. Bove propose a dataflow model for parallel media processing [1]. L. Davis et al develop a multi-perspective video system in University of Maryland [4]. Jason Fritts et al evaluate the characteristics of multimedia applications for media processors [7]. Multiple cameras are deployed in a FlyCam system to capture panoramic video with the cooperation of the cameras [2]. Real time human tracking systems have also been proposed in literature. A real time person tracking system, Pfinder, developed at MIT media lab [14] uses Maximum A Posteriori Probability (MAP) approach to detect and track people by using 2D models. The Pfinder system works in an arbitrarily complex but single person, fixed camera environment. W4 is another real

time human tracking system developed at University of Maryland. It has the ability to track multiple persons in the scene. It also assumes fixed camera environment. The background information should be collected before the system can track foreground objects. A more detail introduction of work concerning human tracking is given by A. Pentland [3].

In our work, we select and combine several tasks in different image and video processing domain to develop a real time video surveillance system. This system has not only the abilities to detect and tracking people, but also the ability to recognize their activities in real time.

The rest of the paper is organized as follows. Section 2 describes the current smart camera system. Section 3 discusses the structure of a more advanced system being developed. Conclusions are given in Section 4.

2 Design of the Prototype System

The smart cameras are used in a fixed environment where the background and illumination do not change over time and where the cameras are also fixed. While this reduces the complexity of the processing algorithm, it brings in the requirement of using multiple camera nodes to overcome the object occlusion and enlarge the sensing field of the whole system. As a real time system, the processing ability is critical. Considering this, each camera node has its own processing units. The camera nodes can communicate with each other to identify the node having the best view angle. Figure 1 shows the architecture of the smart camera system.

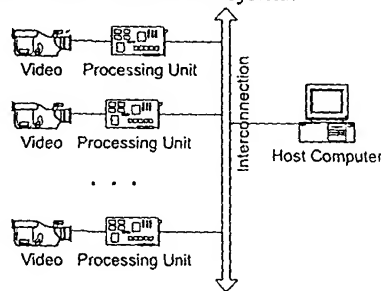


Figure 1: Structure of the whole system

In our prototype system, two cameras, oriented at 90 degrees to each other, are used. The camera nodes, hosted by a PC, can communicate with each other via shared memory.

2.1 Hardware configuration for the prototype system

In our prototype system, a single smart camera node is composed of a standard camera and a video processing board. We use one TriMedia Media processing board [18] in each camera node. A TriMedia board has one TM1300 TriMedia processor, which is specialized for media processing that allows Windows and Macintosh platforms to take advantage of the TriMedia Processor via PCI interface. Multiple TriMedia processing boards can be installed to one host PC to form a more powerful multi-processor system. A 32-bit TM1300 processor has its own dedicated memory and a five issue VLIW (Very Long Instruction Word) CPU together with several coprocessors as shown in Figure 2. The video input and video output units provide a convenient interface for video I/O stream. After initialization, the Video In unit automatically stores the input image into an assigned buffer. When the loading of one frame is finished, an interrupt is arisen to notify the CPU. The CPU can then copy the data in the buffer and supply empty buffers for new frames. Video Out unit puts the frames in the supplied buffers to an output video stream when a frame is processed, the CPU is notified by an interrupt. The frame based

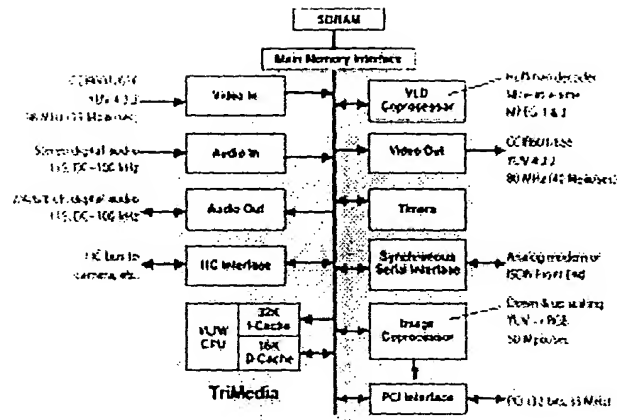


Figure 2: Structure of a TriMedia processor

	Units	#Units	Latency(Cycles)
Functional Unit	Constant	5	1
	Integer ALU	5	1
	Load/Store	2	3
	DSP ALU	2	2
	DSPMUL	2	3
	Shifter	2	1
	Branch	3	3
	Int/Float MUL	2	3
	Float ALU	2	3
	Float Compare	1	1
	Float sqrt/div	1	17
	#Register	128	
Instruction cache		32KB, 8 way	
Data cache		16KB, 8 way	
#Operation slots/instruction		5	

Table 1: Features of TriMedia processors

interrupt scheme reduces the processing load of the CPU.

The CPU in the processor has multiple functional units and 128 registers. Table 1 displays the major features of a TriMedia CPU. In our system, TM1300 processors run at 100MHz, providing a peak performance of 500 MOPS.

2.2 Algorithm and software architecture

Algorithm used in the system is a major factor of system architecture. In this section, we give a brief introduction to the algorithms used in the system. However, since the main focus of this paper is the design of the system, we do not give in-depth description. More details of the algorithm is presented in [5], [6], and [21]. The algorithm consists of two parts, low and high-level processing. The low-level part performs human detection and extracts parameters for the

abstract graph representation of the image being processed. The high level part uses HMM algorithm to determine the movements of the person's body parts, and uses a distance classifier to detect specific gestures. During these processes, each smart camera node communicates with other nodes, so that the node with the frontal view of the person, yields the final results.

The algorithm blocks of the low-level parts are shown in Figure 3. Figure 4 displays a sample frame processed by the algorithm.

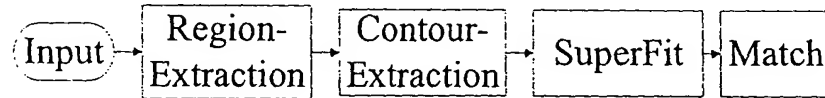


Figure 3: Algorithm blocks for low level processing

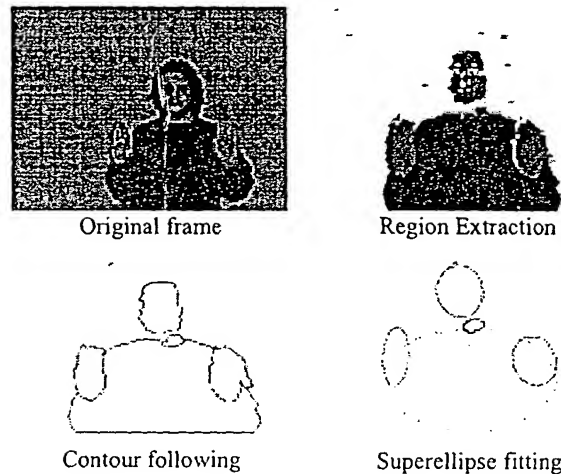


Figure 4: A frame processed by target program

2.2.1 Low-level processing

Region extraction

Region extraction block performs background elimination and skin area detection simultaneously. The output of this block is a frame size buffer and is fed into the next algorithm block.

The background elimination takes 384x240 images as input and extracts foreground objects from the input images. At the same time, skin color detection is performed. Since smart cameras are supposed to operate in a relatively fixed environment, we require that background is known a priori and does not change over time. Hence, the pixel level process simply subtracts background from foreground image. Skin area detection identifies skin regions in the input image by comparing color values of each pixel to a human skin color model. Considering the processing time limit of a real application, we use YUV color model instead of more complex model that employs Farnsworth nonlinear transformation in the algorithm.

Contour extraction:

Contour following uses a 3x3 filter to extract the boundary of each object region obtained by previous block.

Ellipse fitting

In this step, the algorithm finds the ellipse parameters which can optimally describe the boundaries extracted.

Graph matching

The binary features between two regions and the unary features of each region are extracted. These features are fed into the graph matching algorithm to determine the body parts of the person in presence [5].

2.2.2 High-level processing

The high-level algorithm part consists of two subparts. In the first part, the position of each body part, identified by the low-level algorithm, is tracked. HMM algorithm is applied to identify the movement of those parts. In the second part, a distance classifier is employed to combine these movements together to detect the specific gesture made by the person in the scene. While the low-level processing is executed on frame basis, high-level processing takes several consecutive frames to identify the movements.

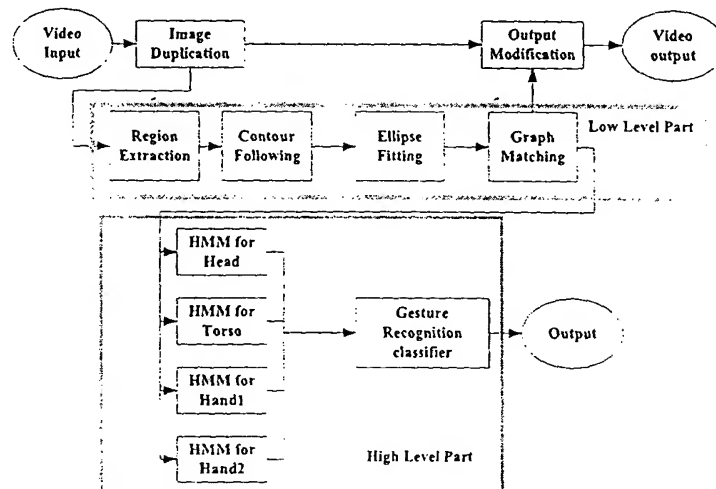


Figure 5: Software architecture

2.2.3 Software Architecture

Software design is a critical part for a real time system. For the prototype system, there are several requirements in addition to the algorithm development.

1. Since the input frame rate is 30 frames/second and the processing capability is varying around 20 frames/sec, there is a need for synchronization.
2. For the demo purpose and for debugging, visualization of the results is needed.
3. The algorithm is still in the developing phase, a flexible software architecture should be used.
4. When the processing capability is concerned, Using of an operation system in a single camera node should be avoided.

Figure 5 shows the software architecture of a smart camera node. In this architecture, we use pipe-filter structure as a basis. By using the interrupt processing ability of the TriMedia processor, we solve the synchronization problem by using the following procedure. A flag variable and a frame buffer are used for video input synchronization. The main processing procedure only accesses the frame buffer when the flag is set to FULL. After the data in the buffer is copied to a safe place, the main processing part sets the flag to EMPTY. The input interrupt-processing procedure only copy data to the frame buffer when the flag is set to EMPTY. If the flag is FULL, the procedure simply discards the current frame. The output synchronization uses a similar scheme. By using this scheme, operation system with multiple thread support is avoided.

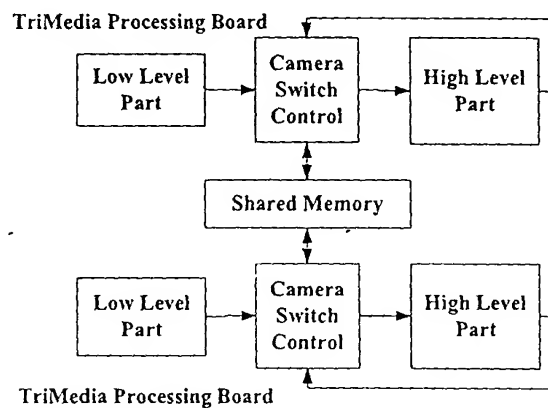


Figure 6: Interconnection between cameras

2.3 Multiple camera coordination

In the prototype system, a simple scheme for multi-camera coordination is implemented (Figure 6). The priority camera switching scheme has two difference levels. The low-level camera switching factor is determined by the comparison between the compactness of the head in two cameras. The high-level switch is triggered by two pre-defined gestures. The high level criteria has priority to low level switch criteria. After the action of camera switch triggered by gestures, the focus is fixed in the active camera for 100 frames before any low-level switch can occur. For each camera node, variables corresponding to the compactness of the head region and the detection of camera switching gesture are calculated and sent into shared memory. Variables from another camera are compared to the local variables to determine if the current camera should be active. In the prototype, only several words are exchanged between two camera nodes for each frame. However, more sophisticated scheme can be adopted for new requirements.

An experiment measuring the communication ability between two camera nodes is conducted. In the experiment, two TriMedia processing boards are configured to exchange frames. It takes 25 seconds to send 10,000 frames from one TriMedia board to another. Each frame contains 92 Kbytes. So the throughput of the shared memory system is 37 Mbytes/second.

Synchronization of the cameras is required in some applications. In the prototype system, the two cameras are not synchronized to each other. But the current system has ability to synchronize the two cameras within one frame, or one thirtieth second.

3 Exploration for the Future System

The goal of the project is to build a smart camera system. We plan to implement the processing unit of a single camera into a single chip. With the advance of the VLIW technology, it is even possible to build the image sensor into the same chip. Therefore, we also conducted experiments to collect information for building a single chip smart camera node.

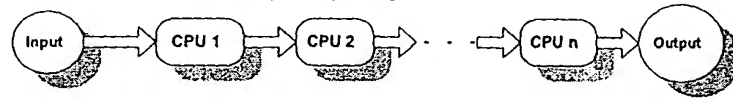


Figure 7: Macro-pipeline architecture

The processing unit can be implemented by using special purpose circuits or by using general CPU-Memory architecture. CPU structure has advantage in its flexibility for the refinement of the system. While in the prototype system, only one processor is deployed in a smart camera node, we also consider using several relative simple processors in a smart camera node. Figure 7 shows a Macro-pipeline architecture model. The pipeline structure is mapped from the pipeline-filter structure of the software. Different algorithm stages can be assigned to different CPUs. A benefit for this approach is that the CPUs can have different capacity. For example, if a CPU is only required to process *region extraction* or *contour following*, the CPU does not need to have floating point unit.

Workload characterization experiments are conducted to determine proper configuration for these CPUs. SimpleScalar tools set is used to conduct the experiment. SimpleScalar tool set is a set of simulators for superscalar microprocessors. Information such as instruction statistics, cache behavior, branch behavior etc can be collected and reported. A detailed instruction for this tool can be find in [19].

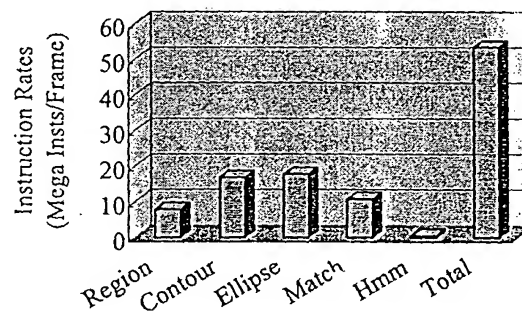


Figure 8: Instruction rates

The first one is the dynamic instruction count of each algorithm block. This is the basis for algorithm block allocation and for determining the object speed of the CPUs. Figure 8 shows the results. The results are counted on frame basis. So the processing capability required by difference blocks are 7.8:16.8:10.7:0.4 or 15:31:33:20:1 approximately, for region:contour:ellipse:match:HMM algorithm blocks, respectively.

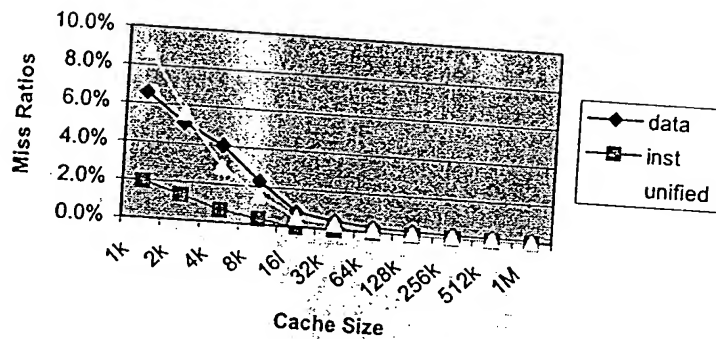


Figure 9: Cache size characteristic of low level algorithm part
(Direct mapped cache, line size 64 bytes)

Another important aspect is the memory system. The cache behavior is also analyzed. Figure 9 shows the effect of different cache sizes for low level algorithm part. The size of a cache is determined by the working set of the program. We define the working set size as the cache size where the miss ratio decreases dramatically with respect to smaller cache size. When such a point does not exist, the smallest cache size resulting the miss ratio below 3% is used. Three classes of cache, data cache, instruction cache and unified cache, are tested. Although our program processes large amount of data, it does not require large data cache. According to the figure, only 8KB cache is required to reduce the data cache miss ratio below 3%. For an instruction cache, a size of 1KB is enough to reduce the miss ratio below 2%. This corresponds to the relatively small size of the program code. Since the instruction cache is small, a unified cache is quite fit to this application. Only 4KB cache can reduce the miss ratio down to 3%. Figure 10 shows the result

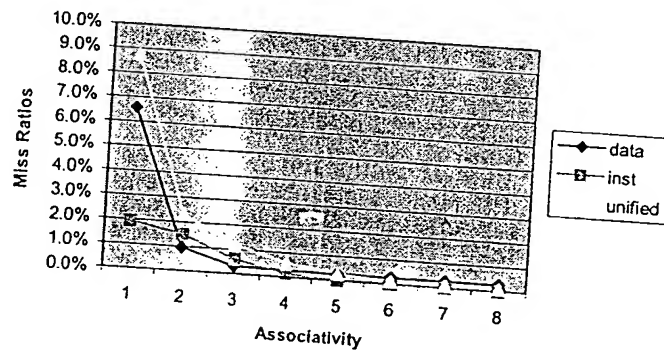


Figure 10: Associativity behavior of low level algorithm part
(Cache size 1kB, line size: 64 bytes)

related to cache associativity. It is clear from the figure that associativity plays an important role in cache behavior. A two-way 1KB data cache outperforms a direct-mapped 8KB data cache. An associativity of 2 or 4 is suitable for the algorithm. In Figure 11, the experimental result concerning cache line size is shown. Although video applications usually process a larger amount of data, this algorithm does not demand larger line size. A line size at 64 bytes is recommended for the algorithm. Figure 12, Figure 13, and Figure 14 display the cache behavior of the high level part. The high level algorithm part has a different data cache behavior from the low level

part. It requires larger data cache size, larger associativity and smaller line size. A possible reason for this is that high level part has a different coding style. The algorithm implementation uses multi-dimensional arrays and uses extensive matrix and vector operations. Since the code size of the high level algorithm part is still kept small, its instruction cache behavior is similar to that of low level algorithm part. Since high level part is not the critical part in terms of running time, the choice of cache parameters largely depends on the cache behavior of low level algorithm. A 1KB four way associative unified cache with 64 bytes cache lines is recommended. We test the effect of such a cache for each algorithm block. The results are shown in Figure 15.

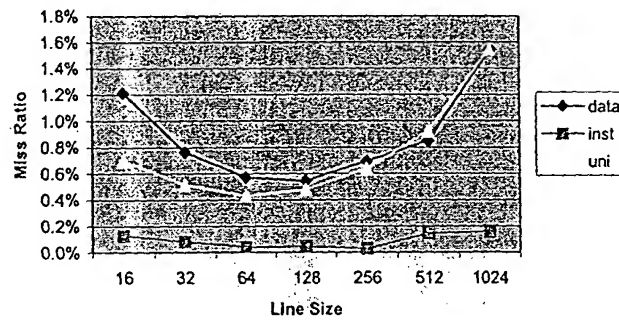


Figure 11: Impact from cache line size for low level algorithm parts
(Cache size: 4kB, associativity:2)

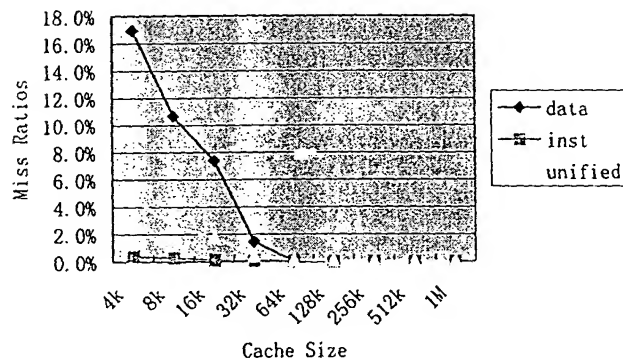


Figure 12: Cache size for high level processing
(Directly mapped cache, line size 64 bytes)

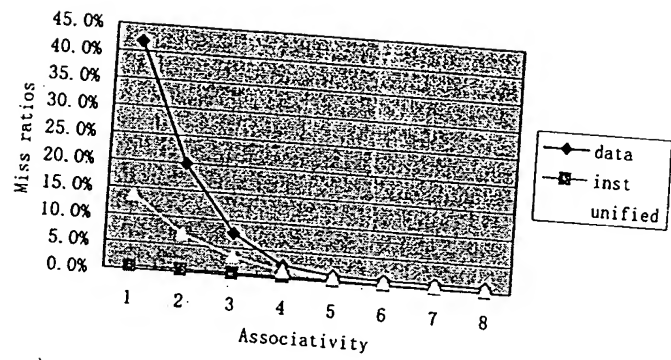


Figure 13: Associativity for high level processing
(1 KB cache, 64 bytes line)

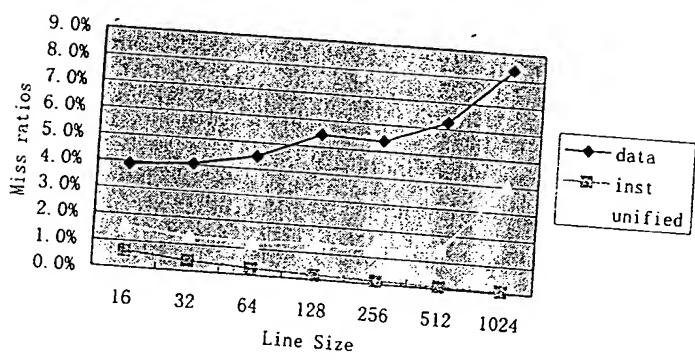


Figure 14: Line size for high level processing

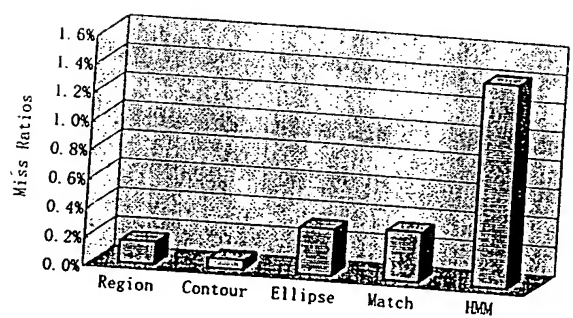


Figure 15: Cache performance for different algorithm blocks

	Execution Time (kcycles)	
	Before optimization	After Optimization
Region Extraction	2737	1168
Contour Following	3145	2070
Whole low level part	7470	4356

Table 2: Effect of special instructions

The impact of media instruction is also tested. A special instruction INONZERO, which performs conditional assignment, is used. Table 2 displays the processing time before and after using this instruction. The results indicate that the instruction is effective to reduce the processing time in low-level pixel processing blocks, namely region extraction and contour following algorithm blocks. Therefore, a similar instruction would be included in the CPU for low level processing in the new system.

4 Conclusions and Future Work

In this paper, we describe a prototype smart camera system developed at Princeton University. Using relatively modest hardware, the system is able to recognize simple gestures of a person in real time. As a step to develop a new hardware system, we also examine the configuration parameters for the new system. Our future work would include developing a new system with improved performance that can recognize more complicated gestures.

References

- [1] J.A. Watlington and V.M. Bove, Jr., "A System for Parallel Media Processing," Parallel Computing, Dec 1997
- [2] J. Foote and D. Kimber, "FlyCam: practical panoramic video and automatic camera control", Proceedings, 2000 International Conference on Multimedia and Expo, IEEE, 2000
- [3] A. Pentland, "Looking at People: Sensing for Ubiquitous and Wearable Computing", IEEE PAMI, Vol 22, No 1, pp. 107-119, Jan. 2000
- [4] L.S. Davis, E. Borovikov, R. Cutler, and T. Horprasert, "Multi-perspective Analysis of Human Action", Third International Workshop on Cooperative Distributed Vision, 1999
- [5] B. Ozer, W. Wolf, A.N. Akansu, "Relational Graph Matching for Human Detection and Posture Recognition", SPIE, Photonic East 2000, Internet Multimedia Management Systems, Boston, MA, November 2000
- [6] B. Ozer, and W. Wolf, "Video Analysis for Smart Rooms", SPIE, ITCOM 2001, Denver, CO, Aug 2001
- [7] J. Fritts, W. Wolf, and B. Liu, "Understanding multimedia application characteristics for designing programmable media processors", SPIE Photonics West, Media Processors '99, San Jose, CA, pages 2-13, Jan 1999
- [8] H. Wu, Q. Chen, and M. Yachida, "Face Detection From Color Images Using a Fuzzy Pattern Matching Method", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 21, No. 6, Jun 1999
- [9] F. Solina, and R. Bajcsy, "Recovery of Parametric Models from Range Images: The Case for Superquadrics with Global Deformations", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 12, no. 2, Feb 1990

- [10] T. Darrel, I. Essa, and A. Pentland, "Task-specific Gesture Analysis in Real-Time Using Interpolated Views", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 18, No.12, pp. 1236-1242, Dec 1996
- [11] J. Yamamoto, J. Ohya, and K. Ishii, "Recognizing Human Actions in Time-Sequential Images Using Hidden Markov Models", Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 379-385, 1992
- [12] T. Starner, J. Weaver, and A. Pentland, "Real-Time American Sign Language Recognition using Desk and Wearable Computer Based Video", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 20, No. 12, pp. 1,371-1,375, Dec 1995
- [13] V. Paviovic, B. Frey, and T. Huang, "Classification Using Mixed State Dynamic Bayesian Networks", Proc IEEE Computer Vision and Pattern Recognition, Vol. 2, pp. 609-615, 1999
- [14] C. Wren, A. Azarbayejani, T. Darrell, A. Pentland, "Pfinder: Real-Time Tracking of the Human Body", Proc. of the SPIE Conference on Integration Issues in Large Commercial Media Delivery Systems, Oct 1995
- [15] I. Haritaoglu, D. Harwood, and L.S. Davis, "A Real Time System for Detecting and Tracking People", 3rd International Conference on Face and Gesture Recognition, Nara, Japan, Apr, 1998
- [16] M. Shaw, D. Garlan, "Software Architecture", Prentice Hall, 1996
- [17] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, "Numerical Recipes in C", Cambridge University Press, Second Edition, 1995
- [18] TriMedia documents, <http://www.trimedia.com>
- [19] D. Burger, and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0", Technical Report 1342, University of Wisconsin Madison, CS Department, Jun 1997
- [20] T. Lv, B. Ozer, and W. Wolf, "Workload Characterization for Smart Cameras", Third Workshop on Media and Streaming Processors, Austin, Texas, Dec 2001
- [21] B. Ozer, T. Lv, and W. Wolf, "Real-Time Video Analysis for Smart Rooms", Submitted to IEEE Trans. Pattern Analysis and Machine Intelligence